**PLANEACIÓN PARA SOLUCIONAR EL MÉTODO 1 Y 2 DEL LABORATORIO 3-INFO 2**

**Método 1:**

1. Crear un archivo txt con un contenido para luego iniciar su proceso de encriptación
2. Utilizar una excepción para verificar que se haya abierto con éxito
3. Contar la cantidad de bytes que tiene el archivo, luego extraigo su contenido en modo binario y verifico que la cantidad de bytes corresponda a la inicial.
4. Cada byte corresponde a un carácter en la tabla ASCII, cabe resaltar que se manejara un archivo con información de cadena, porque puede pasar que haya archivos txt con caracteres no imprimibles de la tabla ASCII, es por eso que se captura la información en modo binario.
5. Entonces ya tengo los datos binarios del texto, a los cuales voy a transformar(encriptar).
6. Por medio de una semilla n (numero entero) voy a separar cada byte que corresponde a 8 bits en grupos de n bits bajo las siguientes condiciones:
7. En el primer bloque, que es el primer byte voy a separar sus bits en grupos de n bits, n debe ser 1<n<8, si el usuario me ingresa (2 o 4) no tendré inconvenientes con la pérdida de bits ya que me quedarían grupos iguales, pero si ingresa (3, 5, 6 o 7) debo rellenar a el último grupo con ceros a la izquierda para no perder bits y tener los grupos parejos. Luego solo en el primer grupo debo invertir todos sus bits. La información binaria extraída la voy a separar en n grupos, cada grupo de bits lo voy a rellenar con 0s a la izquierda para completar los bytes, a el primer grupo le aplico la operación NOT “~” para invertir cada bit.
8. A el siguiente grupo: si hay igual cantidad de 1s y 0s en el grupo anterior aplico NOT ó si hay mayor cantidad de 0s aplico NOT cada 2 bits (4 NOT en total) ó si hay mayor cantidad de 1s aplico NOT cada 3 bits (2 NOT en total porque cada byte tiene 8 bits).

**Solución método 1:**

Inicialmente en el main creo el menú para empezar la manipulación del archivo txt, utilizo un bucle do while para validar la entrada y posteriormente un menú switch para capturar la opción escogida, el menú cuenta con 3 opciones (encriptar archivo, desencriptar archivo y salir), pero solo me enfocaré en la explicación de la primera opción que es la que desarrollé. La primera opción me va a llamar una función que se llama **convertir\_a\_binario** que hace lo siguiente:

1. Pido de una vez la semilla n que debe ser 0<n<8, con el ciclo do while valido que n corresponda.
2. Declaro dos variables, una de tipo string para guardar el nombre del archivo a encriptar y otra de tipo fstream para manipular el archivo en modo lectura, nuevamente con el ciclo do while valido que el nombre del archivo corresponda o sea igual al que está en la carpeta build. El archivo lo abro en modo binario para poder capturar absolutamente todo lo que tiene, luego con las funciones de la librería fstream obtengo el tamaño del archivo(seek, streamsize, tellg).
3. Luego empizo la creación del archivo en modo binario con ofstream y le asigno un nombre (este nuevo archivo lo voy a utilizar más adelante para continuar la transformación), muy importante manejar las expciones en las aperturas de los archivos para no causar errores en la ejecución del programa.
4. Luego creo una variable de tipo char que es la que me va a recorrer carácter a carácter el contenido del archivo original, también creo un arreglo de tipo char del tamaño conocido del archivo original que se obtuvo anteriormente; teniendo esto por medio de un ciclo empiezo a recorrer cada carácter del archivo original y a su vez lo voy convirtiendo a su equivalente en binario por medio de una nueva función llamada **byte\_a\_binario** que me recibe como argumento la variable que me va recorrer cada carácter (se declara como unsigned char para descartar o prevenir char negativos y solo tener los de la tabla ascci de 0 a 255) y el arreglo en donde los voy a almacenar, finalmente después de la conversión a el archivo que tenemos abierto en modo escritura le voy asignar todo lo que esta en el arreglo que son los caracteres pero en modo binario.
5. Ahora explicaré como trabaja la función **byte\_a\_binario:**

Es una función que no retorna nada, pero como la utilizo dentro de una variable voy guardando su resultado en tiempo de ejecución. Esta función me recibe cada byte del archivo original y en un ciclo for que va desde 0 a 7 (tamaño de un byte) voy tomando cada carácter(byte) y lo voy convirtiendo a binario utilizando la siguiente operación (byte & (1 << i)) ? '1' : '0'; donde haciendo AND con ‘1’, pero este un a su vez lo voy moviendo 7 veces para mantener el bit más significativo e ir construyendo el byte, luego como es costumbre en la última posición del byte agrego carácter nulo ‘\0’ y asi se hace sucesivamente con todos los bytes del archivo original.

1. Regresando a el contexto de la función anterior cierro el archivo original y también el nuevo archivo con los caracteres en binario, luego ejecuto una nueva función que se llama **encriptar\_binario**, no necesito validar algo para ejecutarla, ya que con los ciclos do while anteriores ejecutados correctamente el programa continuará con las demás ejecuciones.
2. **encriptar\_binario** hace lo siguiente: es una función que me recibe como argumento la semilla **n**, este número n que fue ingresado por el usuario y validado anteriormente lo guardo en una variable llamada semilla, luego abro el segundo archivo generado en modo binario, al igual que el primero me aseguro que sea abierto correctamente, al igual que el primero obtengo su tamaño, luego creo una variable para recorrer cada bit, reservo memoria para un arreglo que me va a almacenar los 1s y 0s, creo una variable que va a hecer el papel de índice, ya que por medio de un ciclo while mientras el archivo este abierto voy a ir agregando cada bit en el arreglo al cual le reserve memoria, después de completar el arreglo, como se acostumbra con los arreglos tipo char, en su última posición agrego carácter nulo ‘\0’ y el segundo archivo, luego imprimo el contenido del arreglo para compararlo con el segundo archivo (prueba).
3. En esta misma función empiezo el proceso de encriptación: el tamaño del arreglo que es la cantidad de bits lo guardo en una variable, en otra variable guardo la cantidad de grupos que puedo obtener con la semilla n, pero lo redondeo hacia arriba por que si no se forman grupos completos, no puedo desperdiciar los demás bits, ya que también son información.
4. Reservo memoria para un nuevo arreglo que me va a guardar el proceso de encriptación, va ser del tamaño del arreglo anterior más uno para el carácter nulo.
5. Creo dos variables que ve van a guardad la cantidad de unos y ceros, para aplicar las condiciones a los grupos.
6. Utilizo ciclos for anidados: el primer for me marca el recorrido de cada grupo y dentro de este tengo variables que me marcan el inicio y el fin de cada grupo, tambien valido que cuando el recorrido este en el último grupo no vaya a sobrepasar la cantidad de bits reales que tiene el arreglo.
7. Luego tengo una condición que se ejecutará repetidamente después de pasar el primer grupo, ya que solo al primer grupo se le debe invertir todos sus bits, esta condición me va a estar actualizando el conteo de los unos y ceros con respecto al grupo actual, y con base en ese conteo se aplica las debidad operaciones a los grupos siguientes, específicamente en el else de cuando ya no estamos en el primer grupo. Finalmente, después de pasar los primeros ciclos y condiciones en un último for voy guardando el resultado bit a bit, con respecto al grupo actual en el nuevo arregloy como se acostumbra por fuera de todos los for asigno en su posición final el carácter nulo.
8. El resultado de estas transformaciones que ya están en el arreglo las voy a escribir en un nuevo archivo txt, de la misma manera como cree el archivo 2, luego libero la memoria que utilice en los arreglos.
9. Dentro de esa misma función desarrollo el procedimiento para rotar los bits de cada grupo n, una posición a la izquierda.
10. Abro el último archivo generado, obtengo su tamaño, creo variables para recorrerlo, extraer su información y guardar en nuevo arreglo al cual también se le reservó memoria, asi como a los demás se le hizo sus respectivas validaciones para no causar problemas o errores en la ejecución.

**Método 2:**

1. Luego invoco la función para rotar los bits una posición a la izquierda, llamada **rotación\_ izquierda**, que me recibe como argumentos un arreglo con los datos del último archivo generado, su tamaño y la semilla (grupos de n bits), a continuación, explico que hace la función.
2. **rotación\_izquierda:** Es la última función que voy a utilizar para terminar de encriptar el archivo original, esta función no me retorna nada(void), pero en tiempo de ejecución va a hacer lo siguiente:
3. dentro de una variable guardo el resultado de la división del tamaño del arreglo recibido y la semilla, esta vez no tengo necesidad de redondear porque se hizo anteiormente en otra etapa del programa por lo cual ya tengo los grupos reales sin bits sobrantes.
4. Luego reservo memoria para un nuevo arreglo del tamaño del arreglo utilizado en la función anterior, después creo unas variables que me van a servir para ir recorriendo el arreglo que contiene los bits (‘0’ y ‘1’) que vienen del archivo, y los voy agrupando de acuerdo al tamaño n que definio el usuario antes. Para hacerlo, uso una variable llamada valor\_actual, donde voy metiendo los bits uno por uno. Cada vez que llega un bit nuevo, desplazo a la izquierda el número actual (valor\_actual << 1) para hacerle espacio, y luego le agrego el bit nuevo con un OR lógico (|).
5. De esa forma, después de leer n bits, en valor\_actual tengo almacenado ese grupo completo, como si fuera un número en binario.
6. Luego Cuando ya tengo los n bits completos, entonces vuelvo a sacar los bits uno por uno usando desplazamientos hacia la derecha y un AND con 1 para quedarme con cada bit.
7. A cada bit lo convierto de nuevo a carácter '0' o '1', y lo guardo en el arreglo grupos, que es donde tengo todos los grupos armados seguidamente.
8. Por último, uso un contador (indice) para saber en qué grupo voy, y cada vez que completo uno, reinicio el contador y sigo con el siguiente grupo.