**DESAFIO 1**

**INFORMATICA II**

**REALIZADO POR:**

JUAN DIEGO ARBELAEZ MALPICA

SANTIAGO HENAO ZULUAGA

**UNIVERSIDAD DE ANTIOQUIA**

**ANALISIS DEL PROBLEMA**

**CONTEXTUALIZACION:**

* El cliente nos brinda un Dataset que contiene 4 textos planos que fueron comprimidos con (RLE o LZ78) y luego encriptados con (rotación de bits + XOR).
* Nuestras entradas serán:
  + Los mensajes comprimidos y encriptados.
  + Una pista por cada mensaje original.

**OBJETIVOS:**

1. Hallar:
   * Método de compresión usado (RLE o LZ78).
   * Valor de rotación de bits n.
   * Clave XOR K.
2. Desencriptar (desarrollar las operaciones inversas en el orden adecuado ).
3. Descomprimir el mensaje original y compararlo con el fragmento de pista dado.

**TENER EN CUENTA:**

* No utilizar librerías externas que ya resuelvan los métodos de compresión o encriptación.
* Optimizar lo que más se pueda la búsqueda del número de bits (n) y la clave (K) para no probar de forma innecesaria.
* Mantener el orden correcto de las operaciones inversas

**DIAGRAMA DE FLUJO**

Presentamos nuestro primer diagrama de flujo con el que pretendemos llegar a nuestra solución planteada. Teniendo en cuenta que está sujeto a modificaciones que se requieran para poder alcanzar nuestro objetivo.

DESCOMPRIMIR CON RLE

XOR CON K Y ROTAR n BITS A LA DERECHA

K = 0 a 255

0 < n < 8

LEEMOS TEXTO PLANO  
MENSAJE + PISTA

**¿PISTA ENCONTRADA?**

NO

SI

GUARDAR

DESCOMPRIMIR CON LZ78

¿**PISTA ENCONTRADA?**

TEXTO NO UTILIZA NINGUN METODO DE COMPRESION DADO

MOSTRAR TEXTO, n , K

GUARDAR

SI

NO

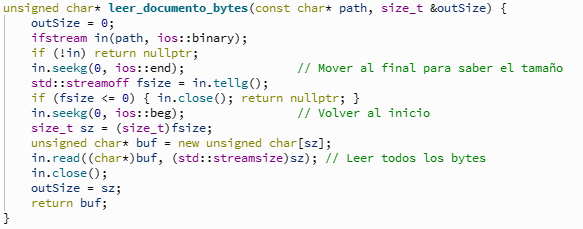
**ALGORITMOS IMPLEMENTADOS**

**Rotación de bits a la derecha:**



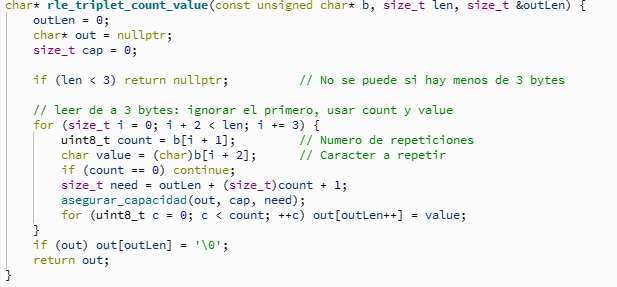
Esta función se encarga de rotar 1 byte n veces a la derecha

**Lectura dinámica de archivos:**



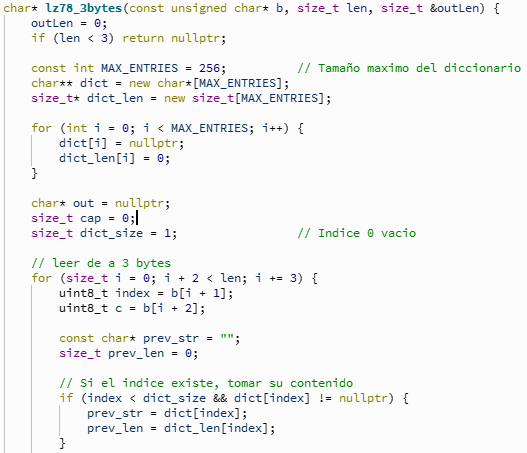
Esta función se encarga de leer los archivos .texto en binarios y devuelve el total de los datos en memoria

**Descompresión RLE:**



Esta función descomprime los datos usando un esquema RLE en este cada bloque está formado por tripletas de 3 bytes.

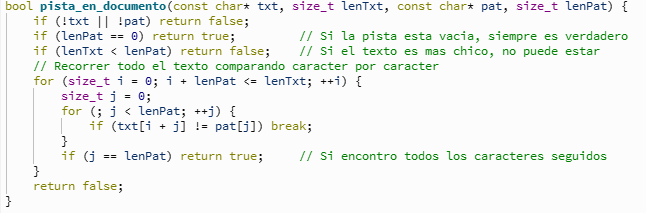
Se obvia el primer byte de cada tripleta. El segundo byte es contador, que indica cuántas veces se repite un carácter. El tercer byte es valor y es el carácter que debe repetirse. Y el resultado es el texto descomprimido almacenado en la memoria dinámica utilizando (char\*).

**Descompresión LZ78:**

Adjuntamos solo una parte del código ya que esta función es un poco mas larga y su objetivo es tomar un arreglo de bytes que representan un texto comprimido con LZ78 y lo reconstruye en un buffer de salida (char\*).El formato tiene la misma dinámica que el anterior y es bloques de 3 bytes:

El primer byte también se ignora, el segundo byte (index) indica una entrada previa en el diccionario y el tercer byte (c) es el nuevo carácter a agregar.

Con esta información, se forma una nueva cadena = (cadena en el diccionario[index]) + c, y se añade a la salida final y al diccionario.

**Búsqueda de pista**:

# Esta función se encarga de verificar si una cadena corta esta contenida dentro de un texto más grande comparando carácter por carácter

# **PROBLEMAS DE DESARROLLO**

**Memoria dinámica:** tuvimos que tener mucho cuidado en las reservas(new[]) y liberaciones (delete[]), para evitar las fugas y tener problemas en el desarrollo de la solución.

**Optimización:** verificar con todas las combinaciones de n y K es costoso ya que son (7 × 256 = 1792 intentos por archivo), pero desarrollamos una verificación temprana con el objetivo de detener la búsqueda en cuanto encontrara la pista.

**Implementación de RLE y LZ78 sin librerías externas:** el desarrollo de estos algoritmos nos complicó un poco ya que no llegábamos al objetivo esperado y decidimos implementarlos sin librerías externas y así pudimos llegar mucho más fácil al objetivo.

**Lectura de archivos con formato desconocido:** llegamos a la conclusión que era necesario limpiar caracteres adicionales (saltos de línea,etc), ya que caíamos mucho en el problema de que la solución nos daba en caracteres no imprimibles.

# **Evolución de la Solución**

* La primera versión entregamos el código como habíamos implementado en el diagrama de flujo, pero nos dimos cuenta que solo funcionaba con unos archivos locales y no con los del problema
* Fuimos desarrollando funciones para la solución que habíamos escrito para llevar un poco más el control
* Montamos los métodos de compresión desarrollados por nosotros mediante los ejemplos entregados para garantizar su funcionamiento
* Desarrollamos los ciclos para poder buscar todas las combinaciones posibles de la clave y el número de rotaciones
* Por ultimo se monta todo el desarrollo que va con la interacción del usuario y así damos por terminado el programa

**Consideraciones**

* Tuvimos que dividir el problema en varias partes (leer archivos, desencriptar, descomprimir, buscar la pista). Esto nos ayudó a contextualizar mucho mejor el problema
* Como nos prohibieron usar string, tuvimos que hacer el desarrollo con punteros y arreglos dinámicos. Esto fue un verdadero reto porque había que cuidar mucho cuándo se reservaba y cuándo se liberaba la memoria para que no se nos acumularan errores.
* Este desafío nos ayudó a practicar el manejo de archivos, punteros, y a entender mejor cómo funcionan los algoritmos de compresión y encriptación. Además, esto nos obligó a ser más ordenados con el código y comentar cada que era posible para ir paso a paso e ir resolviendo el problema mucho más efectivo.