Enunciado:

Resuelva los siguientes ejercicios en C++14 sobre recursión básica. Los ejercicios están, más o menos, en orden de dificultad. Utilice el estándar C++14 en la solución de sus problemas. No olvide compilar con los *flags* apropiados para detectar *warnings* y errores.

1. La empresa RegSeg se dedica al registro seguro de información. Les ha pedido que implementen una estructura ligera, inspirada en el funcionamiento de una block-chain, para almancenar datos de sus clientes. Como su nombre lo indica, una block-chain es una cadena de bloques, que puede recorrerse tanto en una dirección como en otra, pero en donde los bloques se añaden siempre al final. La idea básica es que cada bloque tiene información propia y sobre el bloque que lo precede. Para verificar la información se usa una función que genera un identificador único para el bloque, derivado de los datos que guarda. Si los datos de un bloque se modifican, estos no coinciden mas con el identificador lo que es un indicativo de un cambio no autorizado de la información. Igualmente, si se agrega o borra un bloque de manera no autorizada, la información que tiene el bloque adyacente dejará de coincidir y esto permite detectar el cambio.

   El objetivo es implementar un block-chain simple, usando una lista doblemente enlazada, en la que cadfa bloque tiene un apuntador al bloque que le sigue y otro al bloque que lo precede. Los bloques cumplen con las siguientes características:

   a) La cadena se implementa como una estructura lineal de elementos o bloques, que contienen los siguientes campos: ID, ID del bloque anterior, un String con la información a ser almacenada, un apuntador hacia el siguiente bloque, y un apuntador hacia el bloque anterior. En el caso del primer bloque, el apuntador al bloque anterior será el mismo bloque. En el caso del último bloque el apuntador irá a `nullptr`.

   b) El ID de un bloque se calcula mediante el método `calcID()` que multiplica el valor numérico (entero) de los dos primeros caracteres del String de dicho bloque.

   c) El string de datos del bloque no puede estar vacío, y debe tener una extensión de al menos 2 caracteres.

   d) El método `verifyBlock()` retorna un código numérico que indica si hay un problema con el bloque mismo o su vecino previo.

```
class Block {
public:
  std::string data;
  Block *prev;
  Block *next;
  int ID;
  int ID_prev;

```

```
9     Block(); //Default constructor
10    Block(std::string s); //Constructor that receives data
11
12    int verifyBlock(); //verify the integrity of the block
          and previous neighbour
13    int calcID(); //Compute the ID of the block based on
          its data
14
15    friend std::ostream& operator<<(std::ostream &os, Block
          &b);
16 };
```

Basándose en la plantilla proporcionada, implemente la estructura `Blockchain`, que debe cumplir con las siguientes caracteriticas:

a) Los bloques se añaden al final de la lista usando el método `push()`, que recibe un bloque. Antes de agregarlo se debe verificar que la información del ID del bloque anterior, consignada en el campo del bloque, sea en efecto la que corresponde al contenido del String del bloque anterior. Si este no es el caso, `push()` debe arrojar una excepción con un mensaje de error advirtiendo que el ID no coincide.

b) La cadena debe contar con un método `peek()` que permita ver la información del último bloque.

c) La cadena cuenta con un método `verify()` para verificar la integridad de los bloques, realizando la revisión completa de que el ID de todos los bloques y sus predecesores sea correcto. En caso de que no se cumpla, se debe publicar por pantalla la posición del bloque en que ocurrió el error.

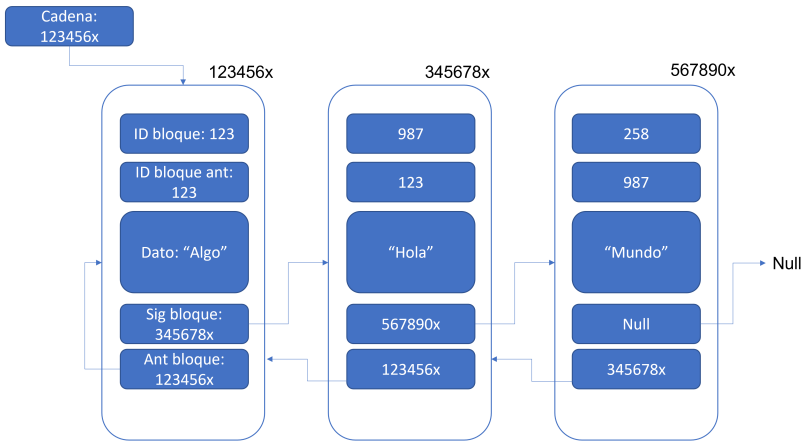

Figura 1: Ejemplo de implementación de la estructura