Índice

1. Introducción

En ciertas aplicaciones es común transferir o almacenar largas secuencias de números.
En algunos casos contamos con que los números están ordenados (como por ejemplo cuando los números representan índices) o están ordenados "localmente".
En este último caso los números no tienen un orden pero presentan la característica que números similares están en posiciones cercanas.

La motivaciones principales y la búsqueda del **Trabajo Práctico 2** son:

- Familiarizarnos con la técnica de programación **RAII** y el lenguaje **C++** en general.
- Familiarizarnos con la programación multithreading y sus buenas prácticas: la detección de regiones críticas que podrían desencadenar *race conditions*, el encapsulamiento de objetos que comparten algún recurso entre distintos threads con *monitors*, la utilización de *condition variables* para sincronizar recursos compartidos entre consumidores y productores, entre otras.

2. Cuestiones de implementación

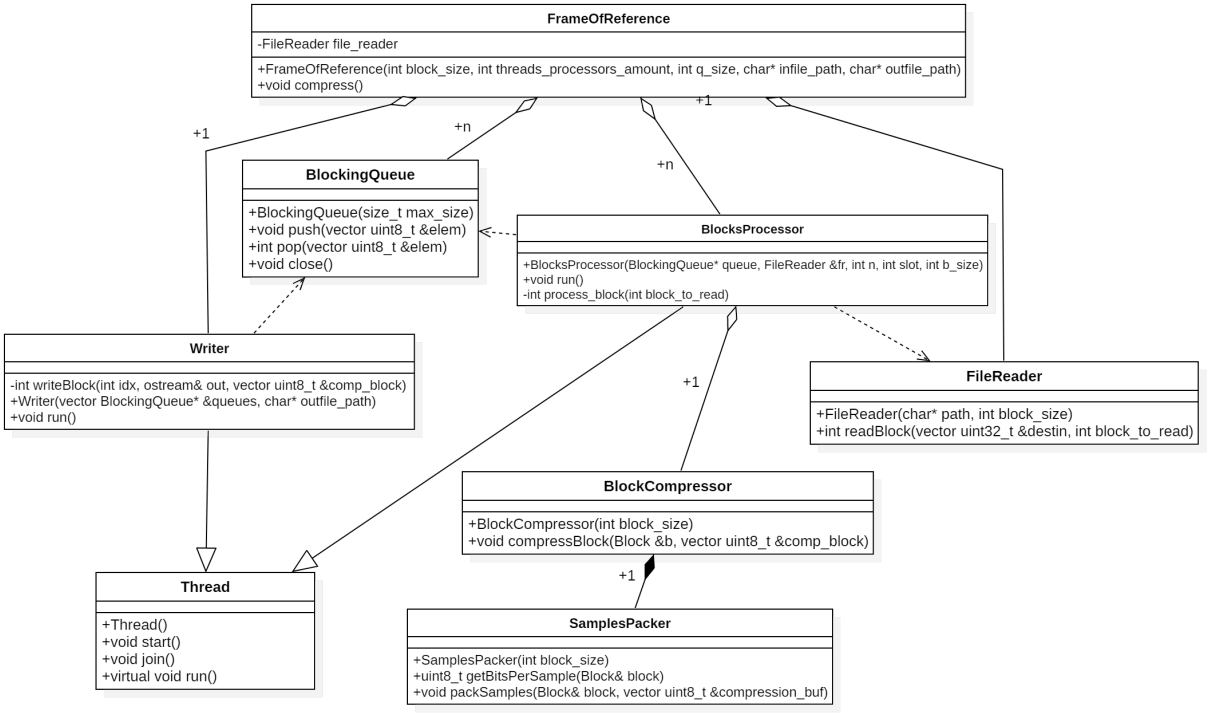
2.1. Descripción de las Clases implementadas

A continuación, mencionaremos y explicaremos brevemente las clases involucradas en la implementación del presente trabajo.

- **BlockingQueue** Es una cola protegida y **bloqueante** diseñada para atender a un consumidor y a un productor. Puede tomar un tamaño máximo y puede cerrarse mediante el método *close*.
- **Block** Contiene un bloque de **n** enteros de 32 bits. Puede obtener el mínimo y el máximo del bloque, restar el mínimo del bloque.
- **SamplesPacker** Empaqueta los bits representativos de un bloque en un vector de enteros sin signo de 8 bits.
- **Block Compressor** Comprime un bloque haciendo uso de un objeto **BlockCompressor** y agregando la referencia y la cantidad de bits utilizados para cada muestra de la compresión.
- **FileReader** Se encarga de leer bloques de memoria, devolviendo mediante distintas constantes señales de fin de archivo.
- **Thread** Es una clase abstracta que contiene en su interfaz a los métodos propios de un *thread* (tales como *run* y *join*).
- **BlocksProcessor** Hereda de la clase **Thread** y se encarga de leer bloques del archivo de entrada (utilizando para ello un **FileReader**), comprimirlos y encolarlos en una cola bloqueante.
- **Writer** Hereda de la clase **Thread** y se encarga de desencolar los bloques comprimidos que *pushean* las distintas instancias de **BlocksProcessor** para escribirlos en el archivo de salida especificado por el usuario.
- **FrameOfReference** Interfaz del compresor. Inicializa los threads y las colas, realiza el *join* de los threads y por último libera la memoria pedida.

3. Diseño del código

3.1. Diagrama de clases



4. Conclusión

4.1. ¿Un posible error en la función seekg?

Llegando a la finalización del presente trabajo, se encontró un posible error a corregir en las librerías de manejo de archivos que proporciona **C++**.
Cuando en la utilización de la función **seekg** nos pasamos del **EOF**, la documentación advierte el encendido del failbit. El problema es que si luego retrocedemos hasta una posición válida, el failbit no se apaga por lo que debemos usar la función **clear** para solucionarlo. Lo que se menciona no es advertido en la documentación y condujo a errores que costaron muchas horas de debuggeo.

4.2. Librerías más importantes utilizadas

1. **bitset** Utilizada en la compresión de los bloques. Fue de gran ayuda a la hora de concatenar la compresión de cada *muestra*[1].]
2. **math** Utilizada en el cálculo de la cantidad de bits a usar en un bloque (cálculo de un logaritmo en base 2).
3. **string** Utilizada principalmente en el empaquetamiento de las *muestras*. Proporciona una gran facilidad para el manejo de los mismos; por ejemplo en el concatenamiento de strings o en la extracción de subcadenas.
4. **queue** Utilizada como base en la implementación de la cola bloqueante.

[1] Entero sin signo de 4 bytes.

5. **condition_variable** Indispensable a la hora de la sincronización de nuestra cola bloqueante.
6. **mutex** Utilizada para proteger los recursos de la cola bloqueante y el lector de archivos.