Fecha: 03/03/2025

Informe Entrega 1 Proyecto

Autores: Santiago Mesa y Jeronimo Chaparro Tenorio.
Materia: Estructura de Datos.
Pontificia Universidad Javeriana

Resumen de lo solicitado:

En la entrega número uno del proyecto se nos solicita, a partir de comandos indicados, hacer la simulación de cargar en memoria una imagen (.ppm o .pgm) o un conjunto de imágenes (llamado volumen) y por medio del conjunto de imágenes generar una proyección en dos dimensiones según una dirección (X, Y o Z) y un criterio (mediana, promedio, minimo o maximo).

Propuesta de solución:

A partir de lo solicitado se propone la creación de dos TADs para el correcto almacenamiento y creación de archivos .ppm o .pgm. Un TAD imagen encargado de almacenar la imagen solicitada, teniendo en cuenta su código (P2 o P3), su nombre, su tamaño en X, su tamaño en Y, su intensidad máxima y todos sus píxeles con sus valores de intensidad. Y un TAD volumen que contenga un conjunto de imágenes, a su vez que su nombre (el del volumen) y su número total de imágenes contenidas.

1. Diseño de TADs: Imagen y Volumen

TAD Imagen:

- **Atributos:**
  - `nombre` (string): Nombre del archivo de la imagen.
  - `codigo` (string): Formato del archivo (por ejemplo, "P2").
  - `xTamano` (int): Ancho de la imagen en píxeles.
  - `yTamano` (int): Alto de la imagen en píxeles.
  - `maxIntensidad` (int): Valor máximo de intensidad de los píxeles.
  - `lista` (list<list<int>>): Lista bidimensional que almacena los valores de los píxeles.
- **Métodos:**
  - `setNombre`, `setCodigo`, `setXTamano`, `setYTamano`, `setMaxIntensidad`, `setLista`: Métodos "set" para establecer los valores de los atributos.
  - `getNombre`, `getCodigo`, `getXTamano`, `getYTamano`, `getMaxIntensidad`, `getLista`: Métodos "get" para obtener los valores de los atributos.
  - `~Imagen()`: Destructor que limpia la lista de píxeles.

TAD Volumen:

- **Atributos:**
  - `nombre` (string): Nombre base del volumen.
  - `nImagenes` (int): Número de imágenes que componen el volumen.
  - `lista` (list<Imagen>): Lista de objetos de tipo `Imagen`.
- **Métodos:**
  - `setNombre`, `setNImagenes`, `setLista`: Métodos "set" para establecer valores de los atributos.
  - `getNombre`, `getNImagenes`, `getLista`: Métodos "get" para obtener los valores de los atributos.
  - `~Volumen()`: Destructor que limpia la lista de imágenes.

2. Comandos y sus Entradas/Salidas

2.1 `cargarImagen(argumentos)`

- **Entrada:**
  - `argumentos[1]`: Nombre del archivo de la imagen.
- **Salida:**
  - Mensaje de error si el archivo no existe o si no puede abrirse.
  - Mensaje confirmando la carga exitosa de la imagen.

2.2 `cargarVolumen(argumentos)`

- **Entrada:**
  - `argumentos[1]`: Nombre base de las imágenes.
  - `argumentos[2]`: Número total de imágenes.
- **Salida:**
  - Mensaje de error si falta alguna imagen.
  - Mensaje confirmando la carga exitosa del volumen.

2.3 `infoImagen()`

- **Entrada:** Ninguna.
- **Salida:**
  - Información sobre la imagen cargada (nombre, ancho, alto).
  - Mensaje de error si no hay una imagen cargada.

2.4 `infoVolumen()`

- **Entrada:** Ninguna.
- **Salida:**
  - Información sobre el volumen cargado (nombre, número de imágenes).
  - Mensaje de error si no hay un volumen cargado.

2.5 `proyeccion2D(argumentos)`

- **Entrada:**
  - `argumentos[1]`: Dirección de proyección (`x`, `y`, `z`).
  - `argumentos[2]`: Criterio de proyección (`minimo`, `maximo`, `promedio`, `mediana`).
  - `argumentos[3]`: Nombre del archivo de salida.
- **Salida:**
  - Mensajes de error si el volumen no está cargado o si los argumentos son incorrectos.
  - Archivo `.pgm` con la proyección 2D generada.
  - Mensaje confirmando la generación y almacenamiento del archivo.

3. Funciones implementadas

3.1: `cargarImagen`: Lee un archivo ppm o pgm, extrae sus dimensiones, codigo y valores de píxeles, y los almacena en un objeto de la clase `Imagen`.

3.2: `cargarVolumen`: Carga múltiples imágenes para formar un volumen, almacena esos objetos `Imagen` en un objeto de la clase `Volumen`.

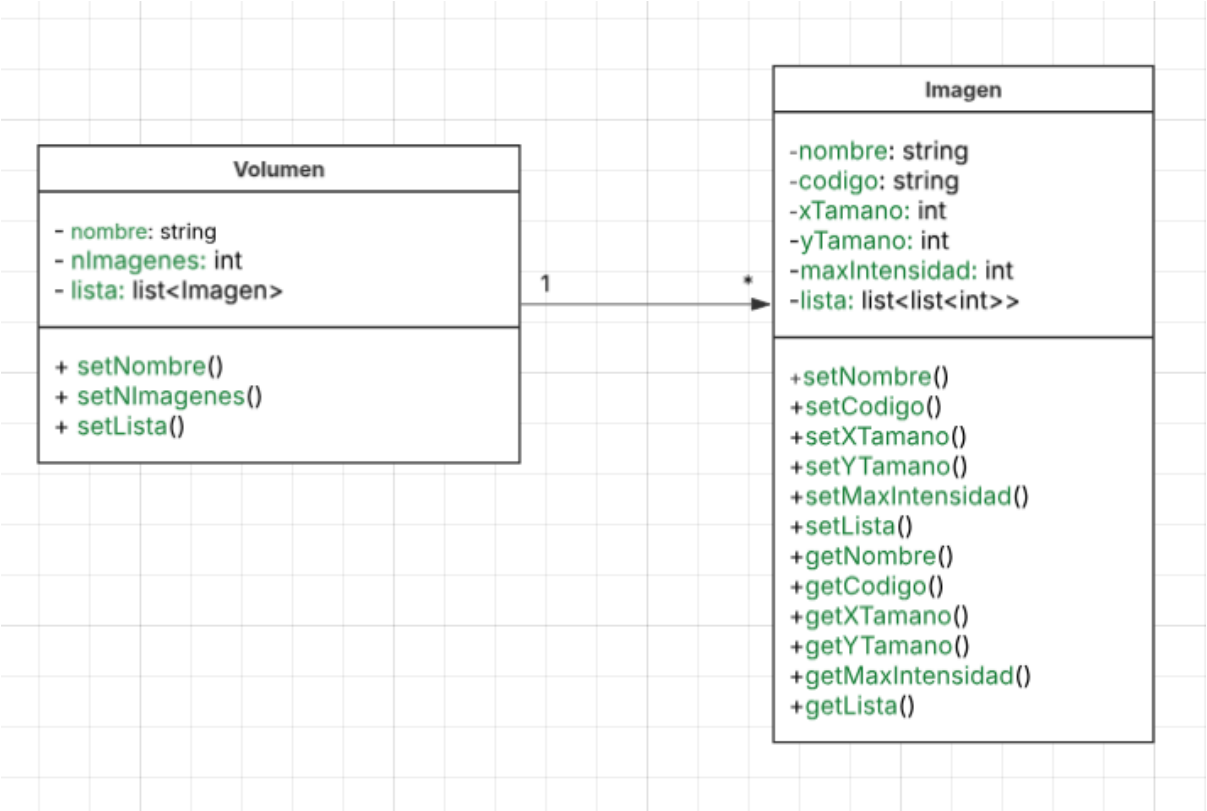
3.3: `infoImagen`: Muestra información básica (nombre, ancho, alto) de la imagen actualmente cargada en memoria.

3.4: `infoVolumen`: Muestra el nombre y el número de imágenes del volumen cargado en memoria.

3.5: `guardarPGM`: Escribe una proyección 2D en un archivo `.pgm`, formateando los valores de píxeles y asegurando la estructura correcta del archivo.

3.6: `proyeccion2D`: Genera una proyección 2D de un volumen en las direcciones `x`, `y` o `z`, usando criterios como mínimo, máximo, promedio o mediana, y guarda la proyección usando `guardarPGM`.

4. Diagrama de Relaciones



5. Plan de Pruebas

5.1 `cargarImagen`

Caso de Prueba	Valores de Entrada	Resultado Esperado	Resultado Obtenido	Estado (Éxito/Falla)
Imagen no existe	**cargar_imagen archivo.pgm**	**“Error: El archivo no existe”**	**“Error: El archivo no existe”**	✅ Éxito
Imagen valida	**cargar_imagen imagenesPrueba/img_02.pgm**	**“La imagen imagenesPrueba/img_02.pgm ha sido cargada.”**	**“Se ha ingresado la imagen: imagenesPrueba/img_02.pgm. La imagen imagenesPrueba/img_02.pgm ha sido cargada.”**	✅ Éxito
Imagen con formato invalido	**cargar_imagen imagenesPrueba/archivo_con_formato_inc**	**“Error: El archivo no existe”**	**“Error: El archivo no existe”**	✅ Éxito

	orrecto.txt			

5.2 `cargarVolumen`

Caso de Prueba	**Valores de Entrada**	**Resultado Esperado**	**Resultado Obtenido**	**Estado (Éxito/Falla)**
mal ingreso de argumentos	**cargar_volumen noexiste 3 mas 3**	**“Error”**	**Error: Uso correcto -> cargar_volumen <nombre_base> <n_im>**	✅ Éxito
Volumen valido	**cargar_volumen imagenesPrueba/t1_icbm_5mm_ppm 3**	**“El volumen t1_icbm_5mm_ppm ha sido cargado.”**	**“La imagen imagenesPrueba/t1_icbm_5mm_ppm/t1_icbm_5mm_01.ppm ha sido cargada. La imagen imagenesPrueba/t1_icbm_5mm_ppm/t1_icbm_5mm_02.ppm ha sido cargada. La imagen imagenesPrueba/t1_icbm_5mm_ppm/t1_icbm_5mm_03.ppm ha sido cargada. El volumen t1_icbm_5mm_ppm ha sido cargado. ”**	✅ Éxito
Numero erroneo de imagenes	**cargar_volumen imagenesPrueba/t1_icbm_5mm_ppm 38**	**“No se pudo cargar el volumen debido a archivos faltantes.”**	**“No se pudo cargar el volumen debido a archivos faltantes.”**	✅ Éxito

5.3 `infoImagen`

Caso de Prueba	Valores de Entrada	Resultado Esperado	Resultado Obtenido	Estado (Éxito/Falla)
Sin imagen cargada	info_imagen	“Error: No hay ninguna imagen cargada en memoria.”	“Error: No hay ninguna imagen cargada en memoria.”	✅ Éxito
con imagen cargada	cargar_imagen imagenesPrueba/img_02.pgm info_imagen	“Imagen cargada en memoria: imagenesPrueba/img_02.pgm, ancho: 423, alto: 500.”	“Imagen cargada en memoria: imagenesPrueba/img_02.pgm, ancho: 423, alto: 500.”	✅ Éxito

5.4 `infoVolumen`

Caso de Prueba	Valores de Entrada	Resultado Esperado	Resultado Obtenido	Estado (Éxito/Falla)
Sin volumen cargado	info_volumen	“Error: No hay ningun volumen cargado en memoria.”	“Error: No hay ningun volumen cargad en memoria.”	✅ Éxito
con volumen cargado	cargar_volumen imagenesPrueba/t1_icbm_5mm_ppm 3 info_volumen	“Volumen cargado en memoria: t1_icbm_5mm_ppm, Tamano: 3.”	“Volumen cargado en memoria: t1_icbm_5mm_ppm, Tamano: 3.”	✅ Éxito

5.5 `proyeccion2D`

Caso de Prueba	Valores de Entrada	Resultado Esperado	Resultado Obtenido	Estado (Éxito/Falla)
Numero incorrecto de argumentos	proyeccion2D x nombre.pgm	“Error: Uso correcto -> proyeccion2D <direccion> <criterio> <nombre_archi	“Error: Uso correcto -> proyeccion2D <direcci ├| n> <criterio> <nombre_archi	✅ Éxito

		vo.pgm>”	**vo.pgm>”**	
volumen no cargado	**proyeccion2D x minimo Xarch.pgm**	**“El volumen aun no ha sido cargado en memoria..”**	**“El volumen a├╢n no ha sido cargado en memoria..”**	✅ Éxito
Direccion invalida	**proyeccion2D arriba minimo Xarch.pgm**	**“Error: Direccion no valida. Use x, y, o z.”**	**“Error: Direcci├│n no v├ílida. Use x, y, o z.”**	✅ Éxito
Criterio invalido	**proyeccion2D x diagonal Xarch.pgm**	**“Error: Criterio no valido. Use minimo, maximo, promedio, o mediana.”**	**“Error: Criterio no v├ílido. Use minimo, maximo, promedio, o mediana.”**	✅ Éxito
Proyeccion Exitosa	**proyeccion2D x minimo Xarch.pgm**	**“La proyeccion 2D del volumen en memoria ha sido generada y almacenada en el archivo Xarch.pgm.”**	**“La proyecci├│n 2D del volumen en memoria ha sido generada y almacenada en el archivo Xarch.pgm.”**	✅ Éxito

6. Guía de compilación

Esta guía explica cómo compilar y ejecutar el proyecto en Linux y Windows utilizando tanto make como g++

1. Compilar y Ejecutar con make

El proyecto incluye un Makefile que permite compilar y ejecutar el programa de manera sencilla. El Makefile es compatible con **Linux** y **Windows.**

En Linux:

Abre una terminal en la carpeta donde se encuentra el *Makefile*.

Compila el proyecto:

$ “make”

Esto generará el archivo ejecutable “***mi_programa.exe”***.

Ejecuta el programa:

***$* “make run”**

Limpiar archivos generados:

***$* “make clean”**

En Windows:

Abre PowerShell en la carpeta donde se encuentra el Makefile (o la terminal de IDE).

Compila el proyecto:

“make”

Esto generará el archivo ejecutable “***mi_programa.exe”***.

Ejecuta el programa:

“make run”

Limpiar archivos generados:

“make clean”

2. Compilar y Ejecutar con *g++*

En Linux:

Abre una terminal en la carpeta src y compila todos los archivos fuente:

$ “g++ -Wall -Wextra -std=c++17 -g main.cpp comandos.cpp interfaz.cpp utilidades.cpp clases.cpp -o mi_programa.exe”

Esto generará el archivo ejecutable **“mi_programa.exe”**.

Ejecuta el programa:

“./mi_programa.exe”

En Windows:

Abre PowerShell en la carpeta srcb (o la terminal de IDE).

Compila todos los archivos fuente:

“g++ -Wall -Wextra -std=c++17 -g main.cpp comandos.cpp interfaz.cpp utilidades.cpp clases.cpp -o mi_programa.exe”

Esto generará el archivo ejecutable **“mi_programa.exe”**.

Ejecuta el programa:

“.\mi_programa.exe”

7. Conclusión

La entrega número uno del proyecto de Estructura de Datos presenta una solución funcional para la carga, almacenamiento y manipulación de imágenes (.ppm y .pgm) y volúmenes, permitiendo generar proyecciones 2D a partir de un conjunto de imágenes. Se destaca el diseño estructurado de dos TADs (Imagen y Volumen), que facilitan la organización y el acceso a la información de las imágenes, así como la correcta implementación de comandos que cubren las funcionalidades requeridas. El plan de pruebas demuestra que el sistema responde adecuadamente ante diversos escenarios, validando tanto el éxito como el manejo de errores.

Sin embargo, es posible realizar ciertas mejoras para optimizar la eficiencia y robustez del programa. En primer lugar, la clase Volumen podría beneficiarse de técnicas de paralelización al momento de generar proyecciones 2D, aprovechando librerías como OpenMP para distribuir el procesamiento entre varios núcleos y reducir los tiempos de ejecución. Por otro lado se podrían implementar validaciones más exhaustivas en la carga de archivos, verificando no sólo la existencia del archivo, sino también su formato y consistencia de datos, para evitar posibles errores durante el procesamiento. También sería recomendable optimizar la manipulación de listas bidimensionales, considerando estructuras de datos más eficientes como vectores planos para acceder a los valores de los píxeles de manera más rápida. Estas mejoras permitirían optimizar el rendimiento y la estabilidad del programa, asegurando una experiencia más eficiente y confiable en el manejo de imágenes y volúmenes.