

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

PROGRAMACIÓN 2
6ta práctica (tipo b)
(Segundo Semestre 2025)

Indicaciones Generales:

- Duración: **110 minutos.**
- No se permite el uso de apuntes de clase, fotocopias ni material impreso.
- No se pueden emplear variables globales. No se puede utilizar la clase **string**. Tampoco se podrán emplear las funciones de C que gesten memoria como **malloc**, **realloc**, **memset**, **strdup** o similares, igualmente no se puede emplear cualquier función contenida en las bibliotecas **stdio.h**, **cstdio** o similares y que puedan estar también definidas en otras bibliotecas. No podrá emplear plantillas en este laboratorio.
- Deberá modular correctamente el proyecto en archivos independientes. Las soluciones deberán desarrollarse bajo un estricto diseño descendente. Cada función no debe sobrepasar las 20 líneas de código aproximadamente. El archivo **main.cpp** solo podrá contener la función **main** de cada proyecto y el código contenido en él solo podrá estar conformado por tareas implementadas como funciones. En el archivo **main.cpp** deberá incluir un comentario en el que coloque claramente su nombre y código, de no hacerlo se le descontará 0.5 puntos en la nota final.
- El código comentado no se calificará. De igual manera no se calificará el código de una función si esta función no es llamada en ninguna parte del proyecto o su llamado está comentado.
- Los programas que presenten errores de sintaxis o de concepto se calificarán en base al 40 % de puntaje de la pregunta. Los que no muestren resultados o que estos no sean coherentes en base al 60 %.
- Se tomará en cuenta en la calificación el uso de comentarios relevantes.
- Se les recuerda que, de acuerdo al reglamento disciplinario de nuestra institución, constituye una falta grave copiar del trabajo realizado por otra persona o cometer plagio.
- No se harán excepciones ante cualquier trasgresión de las indicaciones dadas en la prueba.

Puntaje total: 20 puntos

- La unidad de trabajo será **t:** (Si lo coloca en otra unidad, no se calificará su laboratorio y se le asignará como nota cero). En la unidad de trabajo **t:** colocará el(los) proyecto(s) solicitado(s).
- Cree allí una carpeta con el nombre “**Lab06_2025_2_CO_PA_PN**” donde: **CO** indica: Código del alumno, **PA** indica: Primer Apellido del alumno y **PN** indica: primer nombre. De no colocar este requerimiento se le descontará 3 puntos de la nota final.

Cuestionario

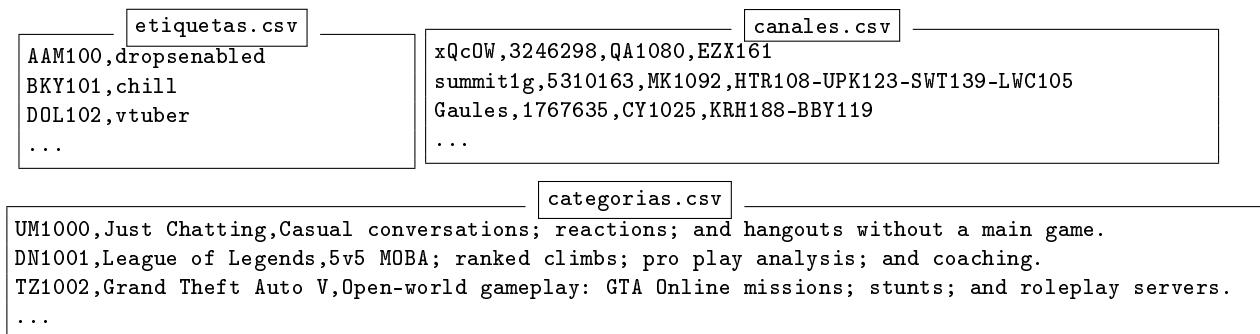
- La finalidad principal de este laboratorio es la de reforzar los conceptos contenidos en el capítulo 3 y 4 del curso: “Programación Orientada a Objetos” y “Operadores Sobrecargados”. En este laboratorio se trabajará con **clases, objetos, constructores, destructores, métodos selectores, métodos completos y operadores sobrecargados**.
- Al finalizar la práctica, comprima la carpeta dada en las indicaciones iniciales empleando el programa **Zip** que viene por defecto en el Windows, no se aceptarán los trabajos compactados con otros programas como **RAR, WinRAR, 7zip** o similares.

Para el diseño de su solución, considere que:

- No podrá emplear arreglos estáticos de más de una dimensión.
- No puede manipular un puntero con más de un índice.
- No puede emplear arreglos auxiliares, estáticos o dinámicos, para guardar los datos de los archivos.
- Los archivos solo se pueden leer una vez.

Descripción del caso

Se cuenta con un proyecto en C++ que buscar gestionar los siguientes archivos de texto: **canales.csv** (nombre, seguidores, código categoria, código(s) de etiquetas), **etiquetas.csv** (código, nombre) y **categorias.csv** (código, nombre, descripción) usando el Paradigma Orientado a Objetos. El proyecto ya se encuentra implementado: incluye las clases con sus atributos, la función **main**, algunos operadores sobrecargados, pero deberá implementar lo que falta para que funcione correctamente.



A continuación se describen cada una de las clases:

Canal: Representa un canal de streaming que contiene información sobre su nombre, categoría, etiquetas y cantidad de seguidores. Además, mantiene un vector de etiquetas asociadas y permite gestionar la información textual de cada una. Se usa para almacenar, copiar y mostrar información de canales, facilitando la carga desde archivos CSV y su manipulación posterior.

- **char *nombre** : nombre del canal.
- **char *categoria** : código o nombre de la categoría a la que pertenece el canal.
- **char *etiquetas** : cadena que contiene los códigos de las etiquetas asociadas.
- **int cantidad_etiquetas** : número total de etiquetas asociadas al canal.
- **char *vector_etiquetas[MAX_VECTOR_ETIQUETAS]** : arreglo de cadenas que almacenan las etiquetas del canal.
- **int seguidores** : cantidad de seguidores del canal.

Categoría: Gestiona la información correspondiente a una categoría, que agrupa canales según su tipo o temática. Cada categoría tiene un código único, un nombre descriptivo y una descripción detallada. Esta clase permite la lectura desde archivos y la comparación con otras categorías.

- **char codigo[7]** : código identificador de la categoría.
- **char *nombre** : nombre de la categoría.
- **char *descripcion** : descripción textual de la categoría.

Etiqueta: Representa una etiqueta o palabra clave asociada a los canales, utilizada para clasificar y filtrar contenido. Contiene un código corto y un nombre que describe la etiqueta, permitiendo también su lectura desde archivos de texto.

- **char codigo[7]** : código identificador de la etiqueta.
- **char *nombre** : nombre o descripción de la etiqueta.

GestorCanales: Actúa como un controlador general encargado de gestionar las colecciones de categorías, etiquetas y canales. Se encarga de la carga de información desde archivos CSV, de enlazar los datos entre las diferentes clases y de presentar las listas ordenadas de categorías, etiquetas y canales.

- `int cantidad_categorias` : número total de categorías cargadas.
- `int cantidad_etiquetas` : número total de etiquetas cargadas.
- `int cantidad_canales` : número total de canales cargados.
- `Categoría vector_categorias[MAX_CATEGORIAS]` : arreglo que almacena los objetos de tipo `Categoría`.
- `Etiqueta vector_etiquetas[MAX_ETIQUETAS]` : arreglo que almacena los objetos de tipo `Etiqueta`.
- `Canal vector_canales[MAX_CANALES]` : arreglo que almacena los objetos de tipo `Canal`.

Pregunta 1 (3 puntos) `GestorCanales::cargar_categorias()`

El método `cargar_categorias` de la clase `GestorCanales` ya se encuentra implementado. No obstante como usted podrá comprobar no compila. Se le pide a Usted que realice las implementaciones necesaria para que el proyecto compile y funcione correctamente. El método `cargar_categorias()` se encarga de leer y almacenar en memoria las categorías disponibles desde un archivo csv. Su función principal es inicializar el conjunto de objetos de tipo `Categoría` dentro del gestor, garantizando que los datos queden listos para ser utilizados por otras operaciones del proyecto.

Restricciones: *No podrá modificar ningún método de la clase `GestorCanales`. No se permitirá la creación de atributos adicionales en ninguna clase. Cada objeto deberá mantener siempre un estado válido y el programa no deberá presentar pérdidas de memoria (memory leaks). Cualquier incumplimiento de estas condiciones será calificado con nota cero.*

Para probar su solución use la función `main` del proyecto, el mensaje `gestor_canales.lista_categorias(cout)`; debería imprimir en pantalla las categorías leídas.

Pregunta 2 (3 puntos) `GestorCanales::cargar_etiquetas()`

El método `cargar_etiquetas` de la clase `GestorCanales` ya se encuentra implementado. No obstante como usted podrá comprobar no compila. Se le pide a Usted que realice las implementaciones necesaria para que el proyecto compile y funcione correctamente. El método `cargar_etiquetas()` se encarga de leer y almacenar en memoria las etiquetas disponibles desde un archivo csv. Su función principal es inicializar el conjunto de objetos de tipo `Etiqueta` dentro del gestor, garantizando que los datos queden listos para ser utilizados por otras operaciones del proyecto.

Restricciones: *No podrá modificar ningún método de la clase `GestorCanales`. No se permitirá la creación de atributos adicionales en ninguna clase. No podrá modificar la sobrecarga `operator>>` que se encuentra en `Etiqueta.cpp`. Cada objeto deberá mantener siempre un estado válido y el programa no deberá presentar pérdidas de memoria (memory leaks). Cualquier incumplimiento de estas condiciones será calificado con nota cero.*

Para probar su solución use la función `main` del proyecto, el mensaje `gestor_canales.lista_etiquetas(cout)`; debería imprimir en pantalla las etiquetas leídas.

Pregunta 3 (3 puntos) `GestorCanales::cargar_canales()`

El método `cargar_canales` de la clase `GestorCanales` ya se encuentra implementado. No obstante como usted podrá comprobar no compila. Se le pide a Usted que realice las implementaciones necesaria para que el proyecto compile y funcione correctamente. El método `cargar_canales()` se encarga de leer y almacenar en memoria los canales disponibles desde un archivo csv. Su función principal es inicializar el conjunto de objetos de tipo `Canal` dentro del gestor, garantizando que los datos queden listos para ser utilizados por otras operaciones del proyecto. Durante esta carga, el atributo `vector_etiquetas` de la clase `Canal` no será cargado.

Restricciones: *No podrá modificar ningún método de la clase `GestorCanales`. No se permitirá la creación de atributos adicionales en ninguna clase. Cada objeto deberá mantener siempre un estado válido*

y el programa no deberá presentar pérdidas de memoria (memory leaks). Cualquier incumplimiento de estas condiciones será calificado con nota cero.

Para probar su solución use la función `main` del proyecto, el mensaje `gestor_canales.lista_canales(cout);` debería imprimir en pantalla los canales leídos.

Pregunta 4 (3 puntos) GestorCanales::operator!()

El método `operator!` de la clase `GestorCanales` ya se encuentra implementado y sobrecarga el operador `!`. No obstante como usted podrá comprobar no compila. Se le pide a Usted que realice las implementaciones necesaria para que el proyecto compile y funcione correctamente. Este operador busca reemplazar en cada canal el código de la categoría por su nombre.

El método `buscar_nombre_categoria` se encarga de localizar el nombre de una categoría a partir de su código identificador. Para ello, recorre el conjunto de categorías cargadas en el gestor y compara cada código almacenado con la llave de búsqueda recibida. Si encuentra una coincidencia, devuelve el nombre correspondiente a dicha categoría; en caso contrario, retorna un valor nulo indicando que no se halló resultado.

Restricciones: *No podrá modificar ningún método de la clase GestorCanales. No se permitirá la creación de atributos adicionales en ninguna clase. Cada objeto deberá mantener siempre un estado válido y el programa no deberá presentar pérdidas de memoria (memory leaks). Cualquier incumplimiento de estas condiciones será calificado con nota cero.*

Pregunta 5 (3 puntos) GestorCanales::operator*()

El método `operator*` de la clase `GestorCanales` ya se encuentra implementado y sobrecarga el operador `*`. No obstante como usted podrá comprobar no compila. Se le pide a Usted que realice las implementaciones necesaria para que el proyecto compile y funcione correctamente. Este operador busca reemplazar en cada canal cada una de los códigos de las etiquetas por su nombre. Las etiquetas las colocará en el vector `etiquetas`.

El método `buscar_nombre_etiqueta` se encarga de encontrar el nombre de una etiqueta a partir de su código. Recorre el conjunto de etiquetas cargadas en el gestor, compara los códigos almacenados con la llave de búsqueda y, si encuentra coincidencia, retorna el nombre asociado a dicha etiqueta. Si no se encuentra el código, devuelve un valor nulo.

El método `agregar_etiqueta` permite añadir una nueva etiqueta al canal actual. Para ello, incrementa el contador de etiquetas y almacena en el vector correspondiente una copia de la cadena recibida. De esta forma, mantiene actualizada la lista de etiquetas asociadas al canal.

Restricciones: *No podrá modificar ningún método de la clase GestorCanales. No se permitirá la creación de atributos adicionales en ninguna clase. Cada objeto deberá mantener siempre un estado válido y el programa no deberá presentar pérdidas de memoria (memory leaks). Cualquier incumplimiento de estas condiciones será calificado con nota cero.*

Pregunta 6 (5 puntos) GestorCanales::operator<<()

Se le pide implementar el operador `operator<<` en la clase `GestorCanales`. El operador de salida `operator<<` tiene la función de mostrar un listado ordenado de los canales.

Para ello, crea una copia local del arreglo de canales con el fin de preservar la información original. A continuación, aplica el algoritmo de ordenamiento rápido (`quicksort`) mediante la función estándar `qsort`, utilizando una función de comparación que evalúa la cantidad de seguidores de cada canal.

En la primera ejecución, el listado se muestra en orden ascendente, es decir, desde el canal con menor número de seguidores hasta el de mayor. En la siguiente llamada, el orden cambia a descendente, y así sucesivamente, alternando entre ambos criterios de visualización en cada impresión.

Finalmente, el operador imprime los diez primeros canales del arreglo ordenado, precedidos por su posición numérica (1→10), incluyendo el nombre, la cantidad de seguidores, la categoría y las etiquetas asociadas a cada canal. El método retorna la referencia al flujo de salida, lo que permite encadenar otras operaciones de salida en el mismo contexto.

Restricciones: *No podrá modificar ningún método de la clase GestorCanales. No se permitirá la creación de atributos adicionales en ninguna clase. Cada objeto deberá mantener siempre un estado válido y el programa no deberá presentar pérdidas de memoria (memory leaks). Cualquier incumplimiento de*

estas condiciones será calificado con nota cero.

Nota: La función `qsort` se utiliza para ordenar arreglos en C y C++ de manera eficiente mediante el algoritmo `quicksort`. Su sintaxis general es:

```
qsort(void *arreglo, size_t tamaño_arreglo, size_t tamaño_del_elemento, int (*función_de_comparacion)(const void*, const void*));
```

Un ejemplo de invocación podría ser: `qsort(vista, MAX_CANALES, sizeof(vista[0]), ascendente);`

Profesores del curso: Miguel Guanira Andrés Melgar
 Rony Cueva Eric Huiza
 Erasmo Gómez

Pando, 31 de octubre de 2025