PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

PROGRAMACIÓN 2
3ra práctica (tipo b)
(Primer 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, ni objetos (con excepción de los elementos de `iostream`, `iomanip` y `fstream`). No se puede utilizar la clase `string`. Tampoco se podrán emplear las funciones de C que gesten memoria como `malloc`, `realloc`, `memset`, `strdup`, `strtok` 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 "Lab02_2025_1_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 2 del curso: "Arreglos y punteros". En este laboratorio se trabajará con **punteros genéricos**, **memoria dinámica** y el método de **asignación de memoria por incrementos**.
- 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

En el contexto de una base de datos, una **tabla** es una estructura que organiza y almacena datos en **filas** y **columnas**, de forma similar a una hoja de cálculo. Cada **tabla** representa una entidad o tipo de objeto sobre el que se desea guardar información. Una **columna** en una tabla de base de datos representa un atributo o característica específica de la entidad que modela esa tabla. Cada **columna** posee un tipo de dato asociado, que define el tipo de valores que se pueden almacenar en esa columna. Las bases de datos suelen manejar varios tipos de datos, pero para efectos de este laboratorio se gestionarán únicamente los siguientes: `INT` para gestionar enteros, `VARCHAR` para gestionar cadenas de caracteres y `DOUBLE` para gestionar números reales.

En este laboratorio se utilizará la estructura `Tabla` para representar una tabla de base de datos. Esta estructura emplea punteros genéricos para manejar dinámicamente tanto las columnas como las filas. La asignación de memoria se realiza utilizando el **método por incrementos**, iniciando con un tamaño de cero columnas y cero filas. La capacidad de columnas se incrementa de dos en dos, mientras que la de filas lo hace de cinco en cinco, conforme se agregan nuevos elementos. Recuerde que no está permitido el uso de constantes mágicas ni de código hardcodeado.

Código 1: `tabla.hpp`

```
struct Tabla{
    void *columnas;
    void *filas;
    int cantidad_columnas;
    int cantidad_filas;
    int capacidad_columnas;
    int capacidad_filas;
};
```

Antes de utilizar la estructura `Tabla`, deberá inicializarla correctamente. Para ello, todos los punteros deben referenciar al valor `nullptr`, y los contadores de filas y columnas deben establecerse en cero.

En esta representación, para cada columna se almacena únicamente el tipo de dato como una cadena de caracteres en mayúsculas. Los tipos de datos se registran en un arreglo donde cada posición corresponde al índice de la columna: la posición 0 contiene el tipo de dato de la primera columna, la posición 1 el de la segunda columna, y así sucesivamente. Por ejemplo, en la Figura 1 se muestra la representación de una tabla que contiene tres campos. El tipo de dato del primer campo es `INT`, el del segundo es `VARCHAR`, y el del tercero es `DOUBLE`. Estos tipos se almacenan como cadenas en mayúsculas, siguiendo el orden de las columnas.

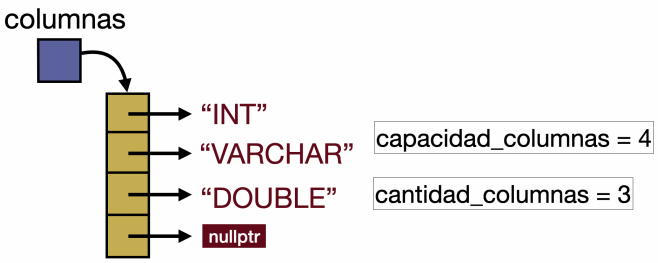

Figura 1: Representación de las columnas de la tabla.

Las filas se almacenan en el campo `filas`, el cual inicialmente apunta a `nullptr`. Al igual que en el caso de las columnas, la inserción de datos se realiza utilizando el método por incrementos. En particular, por cada expansión de capacidad, se reserva un bloque de 5 espacios para almacenar punteros a filas.

En la Figura 2 se muestra la estructura en memoria del campo `filas` luego de insertar una fila. Cada elemento del arreglo de filas es un puntero a un bloque de memoria que representa una fila, y este bloque contiene una cantidad de elementos igual al número de columnas (es decir, al número de tipos de datos definidos en la tabla). Al momento de su creación, todas las referencias dentro de una fila apuntan inicialmente a `nullptr`.

En la Figura 3 se muestra la estructura en memoria del campo `filas` después de insertar datos en una fila. Cada celda de la fila contiene una referencia a un dato almacenado en el `heap` , es decir, un puntero que apunta a una ubicación en memoria dinámica donde se encuentra el valor correspondiente.

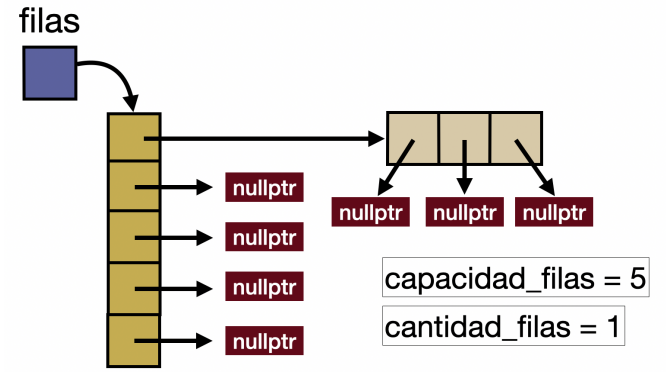

Figura 2: Estructura `fila` luego de una inserción.

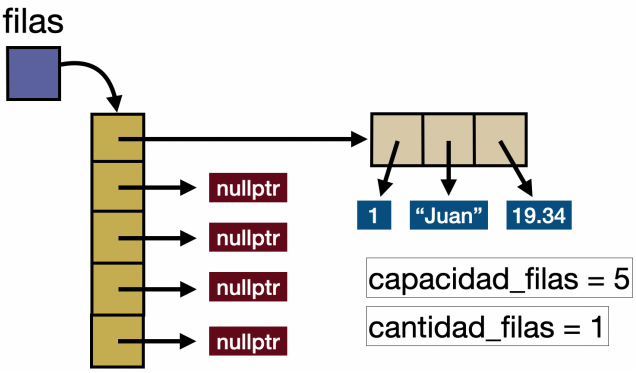

Figura 3: Estructura `fila` inserción de datos.

En la Figura 3 se puede observar la estructura en memoria después de insertar una segunda fila en la tabla. Por su parte, la Figura 4 muestra la misma estructura, pero luego de que se han insertado valores concretos en las celdas de esa fila. Cada celda mantiene una referencia a un dato ubicado en el `heap`, de acuerdo con el tipo de dato definido para cada columna.

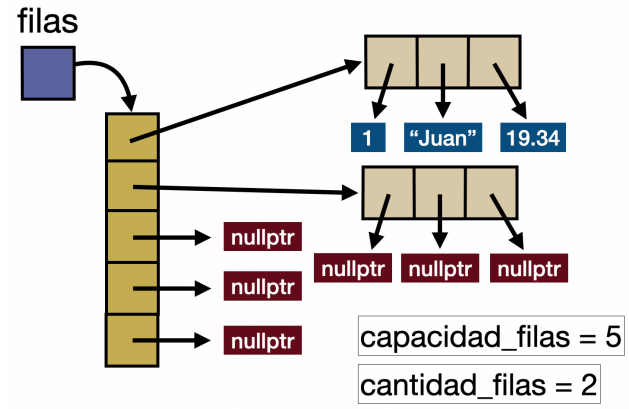

Figura 4: Estructura `fila` segunda inserción.

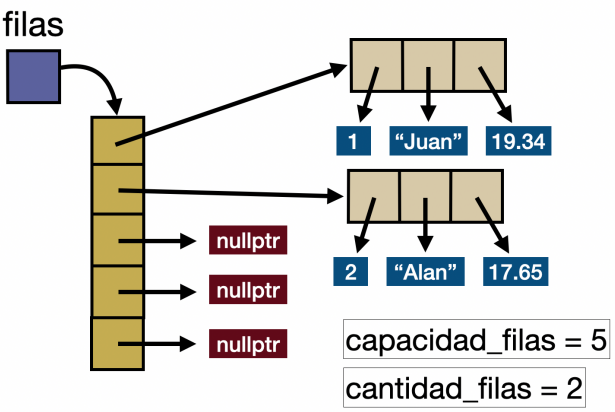

Figura 5: Estructura `fila` segunda inserción de datos.

Pregunta 1

Se requiere implementar lo siguiente, asegurando que la gestión de memoria se lleve a cabo utilizando el método de **asignación por incrementos**:

- (1 punto) Implemente la función `inicializar_tabla`. La función `inicializar_tabla` es responsable de preparar la estructura `Tabla` para su uso, poniéndola en un estado inicial donde no haya datos almacenados, y todos los punteros estén correctamente configurados para evitar referencias a memoria no válida. Su propósito principal es asegurar que la estructura esté limpia y lista para ser utilizada en la inserción y manejo de columnas y filas en el futuro.

- (2 puntos) Implemente la función `insertar_columna`. La función `insertar_columna` tiene como objetivo agregar una nueva columna a la estructura `Tabla` y asegurarse de que haya suficiente espacio en el arreglo de columnas para realizar esta inserción. El tamaño de incremento a usarse es de 2 (ver Figura 1).

- (3 puntos) Implemente la función `insertar_fila`. La función `insertar_fila` tiene como objetivo agregar una nueva fila a la estructura `Tabla`, asegurando que haya suficiente capacidad para ello. Primero, incrementa el contador de filas y verifica si es necesario redimensionar la memoria disponible para las filas. Si es necesario, asigna un nuevo bloque de memoria con el tamaño adecuado. Luego, asigna memoria para la nueva fila, que tendrá tantas posiciones como columnas existen en la entidad. Cada elemento de la fila se inicializa a `nullptr`, lo que garantiza que las celdas estén vacías y listas para almacenar datos posteriormente. Esta función permite gestionar dinámicamente la inserción de filas en una tabla representada en memoria (ver Figuras 2 y 4).
- (2 puntos) Implemente la función `insertar_campo`. La función `insertar_campo` permite almacenar un dato en una posición específica de una fila previamente creada dentro de la entidad. Recibe como parámetros una copia de la entidad, un puntero a la fila, el número de campo (columna) y un puntero genérico referenciando al dato a insertar (ver Figuras 3 y 5).

Pregunta 2

Se cuenta con el archivo `infracciones.csv` que contiene los siguientes campos: código de la infracción (número entero), descripción (cadena de caracteres), tipo (cadena de caracteres) y valor de la multa (real).

`infracciones.csv`

```
101;Adelantar o sobrepasar en forma indebida a otro vehículo.;Grave;316.00
102;No hacer señales ni tomar las precauciones para girar voltear en U.;Grave;316.00
103;Detener el vehículo bruscamente sin motivo.;Grave;316.00
...
```

Se solicita implementar la función `cargar_tabla_infracciones`. La función tiene como objetivo cargar el archivo `infracciones.csv` como una tabla en memoria retornando una estructura `Tabla` conforme descrito anteriormente. Deberá usar las funciones implementadas en la pregunta 1 y realizar lo siguiente:

- (1 punto) Inicializar la estructura `Tabla` conforme indicado anteriormente.
- (1 punto) Insertar las columnas la estructura `Tabla` que sean necesarias para representar los campos del archivo `infracciones.csv`.
- (1 punto) Leer cada uno de los campos del archivo `infracciones.csv`.
- (1 punto) Insertar cada una de las líneas del archivo `infracciones.csv` como una fila de la estructura `Tabla`.
- (1 punto) Insertar en cada una de las filas, los datos correspondientes que se han leído del archivo `infracciones.csv`.

Pregunta 3

En el contexto de bases de datos, un `cursor` es una estructura que permite recorrer fila por fila el resultado de una consulta. Se desea implementar el soporte a cursores en la estructura `Tabla` de forma tal que las filas puedan ser recorridas de la siguiente manera, considerando que `tabla` es una variable del tipo estructura `Tabla`:

Código 2: Recorrido usando cursor

```
void *cursor = abrir_cursor(tabla);
while (hay_siguiente(cursor)) {
    int id = *(int*)obtener_campo(cursor, 1);
    char *nombre = (char*)obtener_campo(cursor, 2);
    double nota = *(double*)obtener_campo(cursor, 3);
    cout<<id<<";"<<nombre<<";"<<nota<<endl;
}
```

Se requiere implementar lo siguiente:

- (2 puntos) Implemente la función `abrir_cursor`. La función `abrir_cursor` recibe una estructura `Tabla` y retorna un puntero al bloque de filas si la tabla contiene al menos una fila; en caso contrario, retorna `nullptr`.
- (2 puntos) Implemente la función `hay_siguiente`. La función `hay_siguiente` permite recorrer secuencialmente las filas de la `tabla`. Recibe como parámetro una referencia a un puntero genérico que representa la posición actual del cursor sobre el bloque de filas. En su primera invocación, la función no avanza el cursor, simplemente valida si apunta a una posición válida. A partir de la segunda llamada, incrementa el puntero para avanzar a la siguiente fila. Si al avanzar el cursor se detecta que la nueva posición apunta a `nullptr`, significa que no existen más filas por recorrer y se retorna `false`; en caso contrario, se retorna `true`, permitiendo continuar con el procesamiento fila por fila.
- (2 puntos) Implemente la función `obtener_campo`. La función `obtener_campo` permite acceder a un campo específico dentro de la fila señalada por el cursor, según el número de campo indicado.
- (1 punto) Recorra la tabla en memoria usando el soporte a cursores implementado usando para ello la tabla cargada en la pregunta 2.

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

Pando, 2 de mayo de 2025