TRABAJO PRÁCTICO

El videoclub "Cinefilia" maneja exclusivos títulos de películas. Con el objetivo de sumar a una plataforma de streaming los títulos y miembros que componen el videoclub, se solicita, primeramente, gestionar los títulos y miembros del videoclub.

Primera parte

Para ello, se nos solicita construir una aplicación que genere un archivo binario (miembros-VC-fechaProceso.dat, ejemplo miembros-VC-20250301.dat) a partir de uno de texto (miembros-VC.txt) que tiene registros de longitud variable. Se debe validar la información de cada uno de los campos según lo detallado más abajo(cuadro). Esta validación debe hacerse una vez convertido todo el registro. Si la validación de alguno de los campos es errónea, el registro entero se descarta y deberá grabarse en un archivo de error llamado error-VC-fechaProceso.txt (error-VC-20250301.txt), junto con el motivo por el cual se descartó (validación).

Al comenzar, debe ingresarse la fecha de proceso y validarla. También deberá permitir elegir entre recuperar archivos modificados en corridas previas a la fecha del proceso (siempre se leerá por defecto el último archivo generado) y el original (miembros-VC.dat). Si existe un archivo para recuperar no deberá procesar el archivo de texto nuevamente.

Estructura de información miembros-VC que se desea tratar:

Campo	Tipo	Validación
DNI (clave)	long	1000000< DNI <100000000
Apellidos y Nombres	char (60)	* Ver más abajo
Fecha de Nacimiento	t_fecha	Validación formal y < fecha de proceso – 10 años
Sexo	char	‘F’ (Femenino), ‘M’ (Masculino).
Fecha de afiliación	t_fecha	Validación formal, <= fecha de proceso y > fecha nacimiento
Categoría	char (10)	‘MENOR’, menor de 18 años ‘ADULTO’, mayor de 18 años.
Fecha de última cuota paga	t_fecha	> fecha de afiliación y <= fecha de proceso.
Estado	char	‘A’ Alta, ‘B’ Baja - Se crean con 'A'.
Plan	Char (10)	'BASIC', 'PREMIUM', 'VIP', 'FAMILY'.
Email Tutor	Char (30)	Si el miembro es menor de edad, deberá tener un tutor asociado. pepe@galleta.com Se deberá validar el formato del email.

Los apellidos y nombres deberán normalizarse de la siguiente forma:

Deben comenzar con letra mayúscula y luego continuar con minúscula.

- Deben estar separado/s del/los nombre/s por una coma. De no existir dicha coma, agregarla a continuación de la primera palabra.
- Cada palabra debe separarse por no más de un espacio.
  ejemplo: galletA pePE raul →Galleta, Pepe Raul

Segunda parte

Crear un menú que permita actualizar el archivo binario miembros-VC-fechaProceso.dat, generado en el punto anterior que contenga las siguientes opciones:

a. Alta
b. Baja
c. Modificación
d. Mostrar información de un miembro
e. Listado de miembros ordenados por DNI
f. Listado de todos los miembros agrupados por plan
g. Salir

Para esto deberá valerse del uso de un TDA Índice implementado en un array con **asignación dinámica de memoria**.

Alta: se obtendrán los datos del teclado, ingresando primero el DNI verificando que no exista en el índice y, cuando estén todos los datos ingresados, se realizará la validación y consistencia de estos (ídem proceso de generación del archivo). Si se detectan errores, se ignora todo lo ingresado. Una vez aceptado, grabarlo al final del archivo binario e insertar la información correspondiente en el índice.

Baja: Si existe la clave que se quiere dar de baja, actualizar el registro correspondiente con un carácter 'B' en estado. Eliminar registro correspondiente del índice.

Modificación: Si existe la clave que se quiere modificar, actualizar la información deseada. Si se detectan errores, se ignora todo lo ingresado.

Mostrar información de un socio: Si existe la clave, mostrar toda la información correspondiente al socio.

Listado de socios ordenados por clave (DNI): Debe mostrar todos los registros

activos (no están dados de baja) ordenados por DNI.

Listado de todos los miembros por plan: Debe mostrar los miembros activos agrupados por Plan y a su vez ordenados por DNI.

Salir: Debe terminar la ejecución del programa.

Como procedimiento final, debe liberar la memoria del índice.

No debe olvidar que para aprobar el trabajo práctico deberá:

El trabajo práctico debe ser modularizado y eficiente según lineamientos de la cátedra, creando bibliotecas y armando macros cuando considere necesario, así como también con un estilo defenido al nombrar variables claras, identación, etc.

Para que el trabajo práctico sea admitido para su corrección deberá estar presentado en tiempo y forma, es decir en la fecha indicada, mediante la plataforma miel y la hora del upload de la herramienta deberá ser antes de las 23:59:59.

Un solo integrante deberá entregar el trabajo práctico a través de la plataforma MIeL, indicando al momento de la entrega los integrantes y el grupo al que representa.

La devolución de los trabajos prácticos será enviada por los docentes por medio de la plataforma MIeL a cada uno de los miembros.

Una vez aprobado el trabajo práctico grupal se evaluará en forma individual una defensa de este en máquina, en forma presencial. La defensa individual del trabajo práctico es condición requerida para la aprobación de la materia.

TDA Índice

Construir un TDA índice en los archivos indice.h e indice.c.

El desarrollo de ambos debe respetar la siguiente estructura:

```c
///indice.h

#ifndef INDICE_H_INCLUDED
#define INDICE_H_INCLUDED

#define CANTIDAD_ELEMENTOS 100
#define INCREMENTO 1.3
#define OK 1
#define ERROR 0
#define NO_EXISTE -1

typedef struct
{
    unsigned nro_reg;
    long dni;
}t_reg_indice;

typedef struct
{
    void *vindice;
    unsigned cantidad_elementos_actual;
    unsigned cantidad_elementos_maxima;
}t_indice;

/*****************************************************************
Descripción:    toma memoria para 100 elementos e inicializa la estructura va-
cía.
Parámetros:     indice: TDA índice.
                nmemb: cantidad de elementos del índice.
                tamanyo: el espacio en bytes ocupado por cada elemento.
Retorno:        n/a.
Observaciones:
*****************************************************************/
void indice_crear(t_indice *indice, size_t nmemb, size_t tamanyo);
```

```
/*****************************************************************
Descripción:    redimensiona el tamaño del índice.
Parámetros:     indice: TDA índice.
                nmemb: cantidad de elementos del índice.
                tamanyo: el espacio en bytes ocupado por cada elemento.
Retorno:        n/a.
Observaciones:  Debe proporcionar el nmemb incrementado en un 30%
*****************************************************************/
void indice_redimensionar(t_indice *indice, size_t nmemb, size_t tamanyo);

/*****************************************************************
Descripción:   inserta en orden según la clave.
Parámetros:     indice: TDA índice.
                registro: el nuevo elemento a insertar en el índice.
                tamanyo: el espacio en bytes ocupado por el elemento a inser-
tar.
                cmp: función de comparación provista.
Retorno:        OK si la operación fue exitosa y ERROR en caso contrario.
Observaciones:  Si el array está lleno, toma un 30 % más de memoria.
*****************************************************************/
int indice_insertar (t_indice *indice, const void *registro, size_t tamanyo,
int (*cmp)(const void *, const void *));

/*****************************************************************
Descripción:   elimina el registro del índice.
Parámetros:     indice: TDA índice.
                registro: el elemento a eliminar.
                tamanyo: el espacio en bytes ocupado por el elemento a inser-
tar.
                cmp: función de comparación provista.
Retorno:        OK si la operación fue exitosa y ERROR en caso contrario.
Observaciones:  -
*****************************************************************/
int indice_eliminar(t_indice *indice, const void *registro, size_t tamanyo, int
(*cmp)(const void *, const void *));
```

```
/*****************************************************************
Descripción:    si la clave existe deja el registro en registro.
Parámetros:     indice: TDA índice.
                registro: el elemento a buscar.
                nmemb: cantidad de elementos del índice.
                tamanyo: espacio en bytes ocupado por el elemento a insertar.
                cmp: función de comparación provista.
Retorno:        NO_EXISTE si no existe o si existe, la posición ocupada dentro
del array.
Observaciones:  -
*****************************************************************/
int indice_buscar (const t_indice *indice, const void *registro, size_t nmemb,
size_t tamanyo, int (*cmp)(const void *, const void *));

/*****************************************************************
Descripción:    determina si el índice contiene 0 (cero) elementos.
Parámetros:     indice: TDA índice.
Retorno:        OK si está vacío, cualquier otro valor si no lo está.
Observaciones:  -
*****************************************************************/
int indice_vacio(const t_indice *indice);

/*****************************************************************
descripción:    determina si el índice contiene el tamaño máximo posible.
Parámetros:     indice: TDA índice.
Retorno:        OK si está lleno, cualquier otro valor si no lo está.
Observaciones:  -
*****************************************************************/
int indice_lleno(const t_indice *indice);

/*****************************************************************
Descripción:    deja el índice vacío.
Parámetros:     indice: TDA indice.
Retorno:        No posee.
Observaciones:  -
*****************************************************************/
void indice_vaciar(t_indice* indice);
```

```
/*****************************************************************
Descripción:    Carga el array desde un archivo ordenado.
Parámetros:     path: la ruta al archivo binario.
                indice: TDA índice.
                vreg_ind: vector de elementos dentro del índice.
                tamanyo: el espacio en bytes ocupado por el elemento a inser-
tar.
                cmp: función de comparación provista.
Retorno:        OK si la operación fue exitosa y ERROR en caso contrario.
Observaciones:  -
*****************************************************************/
int indice_cargar(const char* path, t_indice* indice, void *vreg_ind, size_t
tamanyo, int (*cmp)(const void *, const void *));

#endif // INDICE_H_INCLUDED
```