Trabajo Práctico Integrador

Primera parte

Dada la siguiente estructura de información que guarda datos de un alumno de una universidad:

Campo	Tipo	Validación
dni. (Clave)	long	10.000< dni < 100.000.000
apellido y nombre	Cadena de 40	Normalizar *[1]
fecha de nacimiento	t_fecha	Validación formal y < fecha de proceso – 10 años
sexo	char	'F' o 'M'
fecha de ingreso	t_fecha	Validación formal, <= fecha de proceso y > fecha nacimiento
carrera	Cadena de 4	"INF", "ELE", "IND", "ECO", "DER", "ADM", "MED", "EDF", "FIL".
Cantidad de materias aprobadas	int	>= 0
Fecha de aprobación de la última materia	t_fecha	> fecha-ingreso y <= fecha de proceso. Si no se informa, asumir la fecha de ingreso.
estado	char	Se genera en el alta como 'R'.
Fecha de baja	t_fecha	Se genera en el alta como 31/12/9999

Construir 2 funciones para validar los datos:

- La primera que permita ingresar los datos del alumno por teclado, los datos deben ser validados (según lo que indica el cuadro) a medida que se ingresan. Los datos de estado y fecha de baja deben ser completados automáticamente.
- La segunda que valide los datos de una estructura alumno, según lo indicado en el cuadro, que recibe por parámetro. Además del alumno a validar, debe recibir una fecha que indica el momento que fue procesado el registro. La función debe devolver qué campos dieron error *[2].

Construir una función que muestre un menú y retorne una opción válida.
Construir un main que utilice las funciones anteriores.

*[1] Normalización del apellido y nombres, consistente en:

- El/Los apellido/s y nombre/s deben comenzar con letra mayúscula y luego continuar con minúscula.
- El/Los apellido/s 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.
- La cadena no debe tener espacios al inicio, ni al final de la misma.

*[2] Una forma de especificar qué campos dieron error sería a través de un vector de enteros donde cada posición representa un campo, indicando con un 1 si el campo dio error y 0 si es correcto. Otra forma sería a través de un código especial que, por ejemplo, en un registro que tenga solo dni y sexo, si el dni es erróneo devuelva 1, si el sexo es erróneo devuelva 2, pero que si ambos son erróneos devuelva 3 (Nota: revise el funcionamiento del operador & para resolverlo de esta manera).

Segunda parte

Construir un programa que:
Lea un archivo de texto (**xxxxx.txt**), que contiene un listado de alumnos con los campos:

- dni,
- apellido y nombre,
- fecha de nacimiento,
- sexo,
- fecha de ingreso,
- carrera,
- cantidad de materias aprobadas y
- fecha de aprobación de la última materia: Puede no ser informada.

En registros de longitud variable con los campos separados por "|".

Y genere un archivo binario (**xxxxx.dat**) que contenga sólo los registros válidos del archivo de texto (la validación se debe hacer utilizando la función creada para tal fin en la primera parte). Debe generar también un archivo de texto (**xxxxx_errores.txt**) con los registros inválidos indicando, además de toda la información del registro, el/los motivo/s por el cual es inválido.

Al comenzar, debe ingresarse por teclado la fecha de proceso y el nombre del archivo (**xxxxx**) junto con el path (ruta de acceso) en el que se encuentra.

Tercera parte

Construir un TDA ***índice*** (**t_indice**), donde en cada elemento o entrada del mismo (**t_reg_indice**) se almacena el dni del alumno (long dni) y el número de registro (unsigned nro_reg) que le corresponde a ese alumno en el archivo.
Las primitivas del TDA son:

- **void ind_crear (t_indice* ind)**: inicializa la estructura a *indice_vacio*.
- **int ind_insertar (t_indice* ind, const t_reg_indice* reg_ind)**: inserta en orden el registro reg_ind según el dni. **(1)**
- **int ind_eliminar (t_indice* ind, t_reg_indice* reg_ind)**: recibe el dni a eliminar en el parámetro reg_ind y devuelve en reg_ind el dato completo. **(1)**
- **int ind_buscar (const t_indice* ind, t_reg_indice* reg_ind)**: recibe el dni en reg_ind, y, si existe, devuelve en reg_ind el dato completo. **(1)**
- **int ind_cargar (t_indice* ind, const char* path):** Carga el índice a partir de un archivo ordenado, cada registro del archivo debe tener la estructura de t_reg_indice. **(1)**
- **int ind_grabar (const t_indice* ind, const char* path)**: Graba un archivo con el contenido del índice **(1)**
- **void ind_vaciar (t_indice* ind)**: deja el índice en su estado de vacío.
- **int ind_primero (t_indice* ind, t_reg_indice* reg_ind)**: Deja el primer registro de índice en reg_ind; (1)
- **int ind_siguiente (t_indice*, t_reg_indice*)**: Deja el registro siguiente al último entregado en reg_ind. (1)
- **int ind_fin (const t_indice*)**: Devuelve *verdad* si la última operación de acceso secuencial no entregó el registro por haber llagado al fin de la secuencia y falso en caso contrario.
- **void ind_liberar(t_indice*)**: Libera la memoria utilizada por el índice.

Resuelva el TDA en los archivos *indice.c* e *indice.h*.
(1) : Devuelve 1 (uno) si la operación fue exitosa y 0 (cero) en caso contrario.

Elegir una de las implementaciones del TDA Índice alguno de los anexos de este documento.
No es necesario resolver las funciones ind_primero, ind_siguiente, ind_fin en esta parte.

Generar un índice por dni del archivo de alumnos utilizando el TDA recientemente creado. Al finalizar el proceso, debe guardar el índice en un archivo con el nombre **xxxxx.idx** (es decir, igual al nombre del archivo de alumnos, pero con extensión .idx).

El índice debe ser generado en el mismo momento en que se crea el archivo binario, es decir, que debe modificar el programa realizado en la parte 2 para que genere el índice. Entonces cuando termine el programa va a obtener los archivos:

- xxxxx.dat - con la información de los alumnos.
- xxxxx_errores.txt – con los errores de validación.
- xxxxx.idx – con el índice.

Cuarta parte

Construir un nuevo programa que realice diversas operaciones sobre el archivo de alumnos (xxxxx.dat) y utilice el índice creado anteriormente.

Como procedimientos iniciales, el programa debe solicitar el path del archivo binario (.dat) con el que va a trabajar y verificar la existencia del mismo en cuyo caso deberá generar el índice en memoria (*ind_crear* e *ind_cargar*) a partir del archivo de índices (igual nombre y extensión .idx), que debe existir necesariamente y está ordenado ascendentemente por la dni. Si no existiese el archivo, cancelar el proceso. Ingresar la fecha de proceso.

Mostrar un menú con las siguientes opciones:

- *Efectuar mantenimiento* (altas, bajas)
- *Listar los dado de baja*
- *Listarlos en orden (sin los dados de baja)*

Opción *Efectuar mantenimiento*:
Tendrá un submenú (Alta, Baja).

- Alta: se obtendrán los datos del teclado, ingresando primero la clave (DNI) verificando que no exista en el índice y, cuando estén todos ingresados, se realizará la validación y consistencia de los mismos (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 el registro de índice en el índice.
- Baja: Si existe la clave que se quiere dar de baja y ***estado*** no es 'B', actualizar el registro correspondiente con un carácter 'B' en ***estado*** y con la fecha de proceso como ***fecha de baja***. Eliminar registro correspondiente del índice.

Opción *Listar los dado de baja*:
Listar los registros del archivo (.dat) que estén dados de baja (***estado***='B'), en el orden en que están en el mismo.

Opción *Listarlos en orden*:
Listar los registros activos (no están dados de baja) del archivo (.dat) ordenados por dni

Como procedimientos finales,

- Grabar el archivo de índices.
- Liberar la memoria del índice.

TDA Índice (Array Estático)

Implementación del TDA ***índice***, donde cada elemento o entrada del índice se guarda en las posiciones de un array definido estáticamente.
Las primitivas del TDA son:

- **void ind_crear (t_indice* ind)**: inicializa la estructura a *indice_vacio*.
- **int ind_insertar (t_indice* ind, const t_reg_indice*)**: inserta en el array en orden según el dni.
- **int ind_eliminar (t_indice* ind, t_reg_indice*)**: elimina el dni informado del array.
- **int ind_buscar (const t_indice* ind, t_reg_indice* reg_ind)**: si el dni existe deja el registro en reg_ind;
- **int ind_cargar (t_indice* ind, const char* path):** Carga el array desde un archivo ordenado: **(1)**
- **int ind_grabar (const t_indice* ind, const char* path)**: Graba un archivo con el contenido del índice
- **void ind_vaciar (t_indice* ind)**: deja el índice en su estado de vacío.
- **int ind_primero (t_indice* ind, t_reg_indice* reg_ind)**: Deja el primer registro de índice en reg_ind.
- **int ind_siguiente (t_indice* ind, t_reg_indice*)**: Deja el registro siguiente al último entregado en reg_ind.
- **int ind_fin (const t_indice* ind)**: Devuelve *verdad* si la última operación de acceso secuencial no entregó el registro por haber llagado al fin de la secuencia y *falso* en caso contrario.
- **void ind_liberar (t_indice* ind)**: No aplica a la implementación (dejar vacía).

TDA Índice (Array Redimensionable)

Implementación del TDA ***índice***, donde cada elemento o entrada del índice se guarda en las posiciones array creado con memoria dinámica, el cual se va a ser inicializado con un tamaño pequeño y con el uso se va a ir agrandando para guardar todos los elementos que sea necesario almacenar.
Las primitivas del TDA son:

- **void ind_crear (t_indice* ind)**: toma memoria para 10 elementos e inicializa la estructura a *indice_vacio*.
- **int ind_insertar (t_indice* ind, const t_reg_indice* reg_ind)**: inserta en orden según el dni. Si no hay lugar en el array, tomar memoria para un 30 % más de lo que tenía.
- **int ind_eliminar (t_indice* ind, t_reg_indice* reg_ind)**: elimina el dni informado.
- **int ind_buscar (const t_indice* ind, t_reg_indice* reg_ind)**: si el dni existe deja el registro en reg_ind.
- **int ind_cargar (t_indice* ind, const char* path):** Redimensiona el vector al tamaño exacto para almacenar todas las entradas del índice y luego carga el array desde un archivo ordenado.
- **int ind_grabar (const t_indice* ind, const char* path)**: Graba un archivo con el contenido del array.
- **void ind_vaciar (t_indice* ind)**: deja el índice en su estado de vacío.
- **int ind_primero (t_indice* ind, t_reg_indice* reg_ind)**: Deja el primer registro de índice en reg_ind.
- **int ind_siguiente (t_indice* ind, t_reg_indice* reg_ind)**: Deja el registro siguiente al último entregado en reg_ind.
- **int ind_fin (const t_indice* ind)**: Devuelve *verdad* si la última operación de acceso secuencial no entregó el registro por haber llagado al fin de la secuencia y falso en caso contrario.
- **void ind_liberar (t_indice* ind)**: libera la memoria utilizada por el índice.

TDA Índice (Lista Simplemente Enlazada)

Implementación del TDA ***índice***, donde cada elemento o entrada del índice se guarda en los nodos de una lista simplemente enlazada. El índice debe implementar el TDA Lista y utilizar sus primitivas.

Las primitivas del TDA son:

- **void ind_crear (t_indice* ind)**: Crea la lista.
- **int ind_insertar (t_indice* ind, const t_reg_indice* reg_ind)**: inserta en la lista en orden según el dni.
- **int ind_eliminar (t_indice* ind, t_reg_indice* reg_ind)**: elimina el dni informado de la lista y devuelve la entrada completa en reg_ind.
- **int ind_buscar (const t_indice* ind, t_reg_indice* reg_ind)**: si el dni existe en la lista deja el registro en reg_ind.
- **int ind_cargar (t_indice* ind, const char* path):** Carga la lista desde un archivo ordenado.
- **int ind_grabar (const t_indice* ind, const char* path)**: Graba un archivo con el contenido de la lista.
- **void ind_vaciar (t_indice* ind)**: deja el índice en su estado de vacío (vacía la lista).
- **int ind_primero (t_indice* ind, t_reg_indice* reg_ind)**: Deja el primer registro de índice en reg_ind.
- **int ind_siguiente (t_indice* ind, t_reg_indice* reg_ind)**: Deja el registro siguiente al último entregado en reg_ind.
- **int ind_fin (const t_indice* ind)**: Devuelve *verdad* si la última operación de acceso secuencial no entregó el registro por haber llagado al fin de la secuencia y falso en caso contrario.

TDA Indice (Árbol)

Implementación del TDA ***índice***, donde cada elemento o entrada del índice se guarda en los nodos de un árbol. El índice debe implementar el TDA Árbol y utilizar sus primitivas.