TAD: libArchivos.h

En este TAD se encuentran las funciones para el manejo de archivos binarios

abrirArchivoLectura()

Se intenta abrir el archivo db.dat en modo Lectura Binaria. Si el archivo no existe

- Se muestra un mensaje por pantalla indicando que se creara este archivo
- Se crea un archivo en modo escritura binaria, si ya existía, es reemplazado
- Se vuelve a abrir el archivo en modo lectura binaria

Esta función devuelve el puntero al archivo listo para lectura

---

abrirArchivoEscritura()

Se intenta abrir el archivo en modo lectura-escritura binaria. Si el archivo no existe

- Se muestra un mensaje indicando que el archivo se está creando
- Se abre en modo escritura binaria para crearlo
- Se cierra el archivo y se reabre en modo lectura - escritura

Esta función devuelve el puntero al archivo listo para lectura

---

grabarLista()

Se abrirá el archivo en modo de escritura binaria. Seguido a esto, se inicializa un puntero **actual** que apunta al inicio de la lista. Mientras este puntero NO sea nulo, se escribirá el nodo actual usando fwrite. Acto seguido, el puntero avanza al siguiente nodo

---

cargarLista()

Se abre el archivo en modo lectura binaria, además de definirse tres punteros, uno para el inicio de la lista, otro para el elemento más reciente y uno temporal para leer los datos del archivo. Se leen todos los datos del archivo y se van guardando en el puntero temporal. Creamos un nuevo nodo para la lista enlazada y copiamos los datos de la variable temporal, luego se pueden dar 2 casos

- Si es el primer nodo, se agrega al inicio
- En iteraciones siguientes, se enlaza el nuevo nodo al final de la lista

Esta función devuelve una lista enlazada

---

getContactoNum(tString)

Se abre el archivo en modo lectura binaria, además de declarar una variable contacto. Acto seguido se recorre todo el archivo, y en cada iteración se guardan valores en la variable contacto que creamos. Si fread no logra leer un contacto completo o se llegó al final del archivo se detiene la búsqueda. Se utiliza **strcmp** para comparar el campo número del contacto actual con el número recibido por parámetro. En caso de que se encuentre una coincidencia, se reserva memoria para un contactoEncontrado y se copian los datos del contacto actual a esta nueva estructura, retornando el contactoEncontrado y finalizando la función

En caso de no hallar una coincidencia, se reserva memoria para un contactoNoEncontrado, que en su número telefónico se guarda un mensaje de error. Acto seguido se retorna este contacto

TAD: libContactos.h

En este TAD se encuentran las funciones para el manejo de contactos

crearContacto()

Esta función se encarga de crear un nuevo contacto en memoria dinámica con los datos ingresados por el usuario (nombre, apellido, número telefónico y nota). Verifica que el número telefónico sea válido y, si no se ingresa una nota, asigna un mensaje predeterminado.

Primero se guarda la memoria para el nuevo contacto. Seguido a esto, se piden los datos para el contacto que vamos a ingresar usando fgets(), en el caso del número telefónico, se implementa una verificación para asegurarnos que todos los caracteres son valores del 0 al 9, si esto no se cumple, pedimos de nuevo el número

En el caso de la nota, si el usuario no agrega ninguna, se le da un valor por defecto. Por último, al valor siguiente le asignamos NULL, este valor se asignará más tarde. La función retorna el nuevo contacto

En distintas partes de la función se ve la declaración **while (getchar() != '\n')**. Esto lo hacemos para limpiar el buffer de entrada

---

editarContacto(const char)

Esta función permite modificar los datos de un contacto existente en la lista de contactos, identificándose mediante su número telefónico. Actualiza los datos directamente en la lista en memoria y persiste los cambios al archivo mediante la función grabarLista()

Primero se usa la función getContactoNum() para obtener el contacto que coincide con el número telefónico recibido. Se piden los valores a modificar de la misma forma que en la creación del contacto. Luego recorremos la lista de contactos para encontrar el nodo que coincide con el número proporcionado, una vez localizado se actualizan sus valores. Si logramos actualizar el nodo, se vuelve a grabar la lista para asegurar la persistencia de datos, por último liberamos la memoria del nodo a editar

---

eliminarContacto()

Esta función permite eliminar un contacto de la lista, identificándose mediante su número telefónico.

Primero verificamos que la agenda no esté vacía, acto seguido se declaran 2 variables, actual y última. La función recorre la lista comparando el número del contacto actual con el número proporcionado. Si se encuentra, se elimina el nodo y se ajustan los nodos siguientes a este. Por último se libera la memoria del nodo eliminado y grabamos la lista de nuevo.

---

liberarLista()

Esta función se encarga de liberar la memoria de una lista enlazada

Lo primero es declarar una lista para recorrer cada nodo y así eliminarlo. Mientras esta no sea null,

- El puntero actual se asigna al nodo que queremos eliminar
- Se avanza el puntero al siguiente nodo
- Se libera memoria del nodo actual

Esta función se utiliza al mostrar contactos.

---

copiarLista(tContacto)

Esta función recibe una lista y devolverá una copia de esta, esto se hace para asegurarnos no tener errores al manejar memoria dinámica.

Si la lista está vacía, se retorna null, seguido a esto, creamos 3 variables

- Copia: Apuntará al primer nodo de la lista copiada
- Último: Mantiene la referencia del último nodo copiado
- Actual: Puntero que recorre la lista original

Recorremos la lista original, y en cada iteración asignamos memoria para un nuevo nodo, copiamos los datos del nodo actual a este e inicializamos siguiente en NULL.

Si es el primer nodo, se asigna como el primero de la copia, para los nodos posteriores, se actualiza el puntero siguiente para que apunte al nuevo nodo, además de actualizar el último, para hacer referencia al último nodo agregado.

Una vez se copiaron todos los nodos, retornamos la lista copiada

---

mostrarContactos(tContacto)

Esta función imprime los cambios nombre, apellido y número de nuestra agenda. Se usa una copia de la lista para asegurarnos de no modificar en ningún momento la lista original.

Iniciamos una copia de la lista mediante la función explicada antes, acto seguido se imprime un mensaje inicial. Mediante un puntero actual, iteramos sobre cada nodo de la copia, e imprimimos los campos mencionados antes. Por último, se libera la memoria de la copia de la lista

---

mostrarContactoEspecifico(tContacto, tString)

Al igual que mostrarContactos, lo primero que se hace es hacer una copia de la lista, para asegurarnos que la lista original no se verá afectada en ningún momento. Mediante un puntero actual, iteramos sobre cada nodo de la copia, comparando el número telefónico recibido con el número del contacto actual. Si se encuentra una coincidencia, se muestra el nombre, apellido, número y nota, además de establecer una variable booleana en true.

Si no se encuentran coincidencias, la variable booleana mantendrá su valor falso, esto nos ayudará para imprimir un mensaje de error. Por último se libera la memoria de la lista

---

esNumeroValido(const char*)

Esta función se usa como soporte para cargar el número telefónico. Recorre la cadena de caracteres “número” hasta encontrar un valor nulo (que indica el fin de la cadena). Para cada carácter se verifica si es un dígito, usando la función isdigit() de ctype.h

Si encuentra un carácter que no es un dígito, devolverá false