**ARCHIVO :**es un conjunto de bytes y van a estar almacenados en un dispositivo,

generalmente tiene un nombre y contiene información.

Pueden ser:

* archivos que utiliza el sistema operativo
* archivos ejecutables que son los archivos binarios ejecutables que generamos

nosotros

* archivos de datos. Los archivos de datos son los que nuestros ejecutables van a poder eh abrir y leer o escribir para poder guardar la información y posteriormente volverla a leer. La idea es que sirva como una especie de persistencia de lo que está en memoria para después poder en un sesión posterior seguir trabajando con esa información.

Para poder trabajar con archivos de datos hay que tener una serie de reglas

* los registros deben tener una longitud fija ( eso lo vamos a definir nosotros cuando definamos la clase que después vamos a guardar en el disco)
* los registros tienen que ser identificados y ese valor que obviamente los va a identificar es, dentro del archivo, un dato único e irrepetible.

Siempre en lugar de strings vamos a guardar vectores de char, porque el string utiliza memoria dinámica (el tipo de dato string), en cambio un vector de char es una cantidad de caracteres fija.

**Generalmente se llama escribir cuando guardamos en el almacenamiento y leer cuando del almacenamiento lo pasamos a la memoria**

Texto

El contenido generado por IA puede ser incorrecto.

No se trabaja con números de registro sino con cantidad de bytes que se escriben o se leen

Cuando hacemos un fread nosotros le decimos, de este dirección de memoria (&reg ) y del tamaño de la

clase alumno (sizeof(Alumno)), que lea un registro (1) desde un puntero (p), que representa nuestro archivo en el disco bien eso. De esa manera nosotros con este puntero file, que abrimos previamente, estamos apuntando a un almacenamiento en el disco, de ahí lo que hacemos es leer una cantidad de bytes desde ese almacenamiento, en la variable que nosotros le indiquemos.

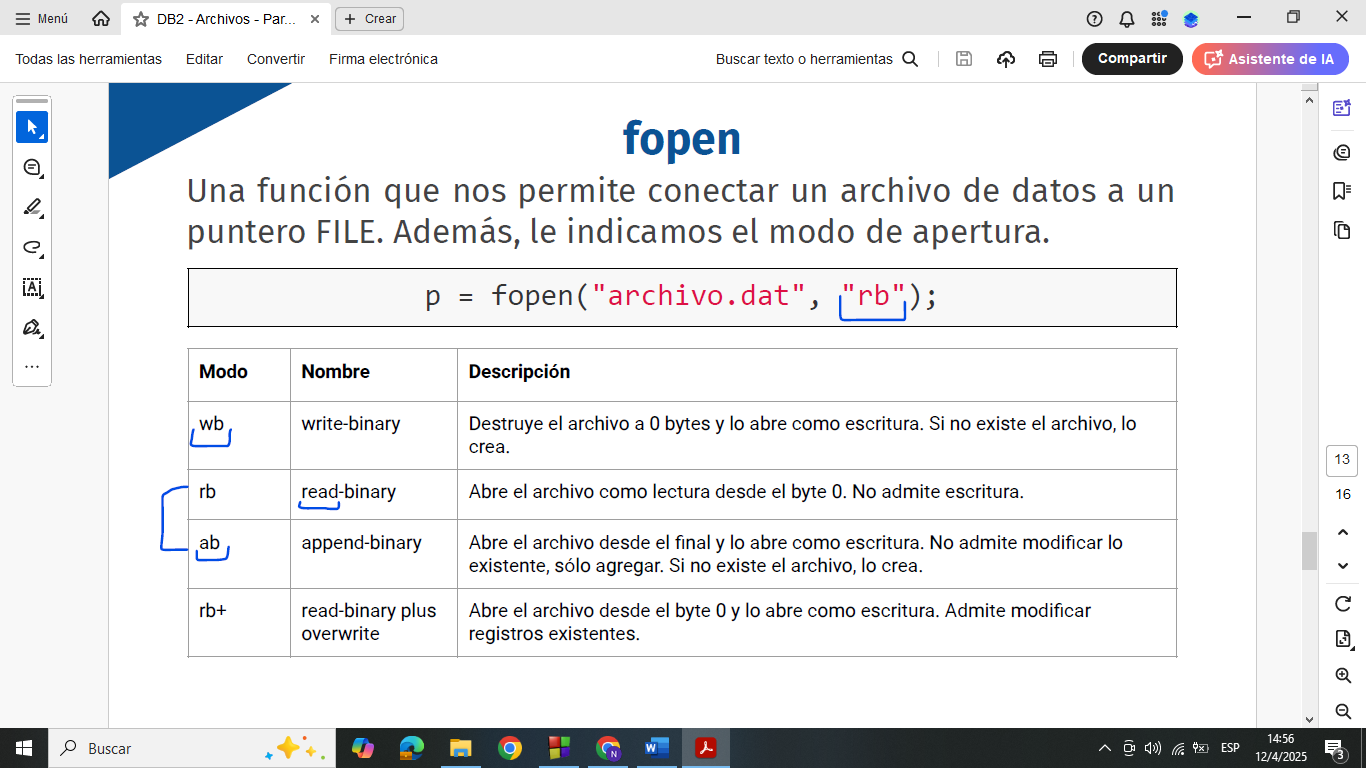
Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

La escritura es el proceso inverso: esta vez es la variable (&reg) la que tiene que contener los datos : Desde esa dirección de memoria (&reg), se escribe en el puntero file (p) una determinada cantidad de bytes (sizeof (Alumno)), un registro (1). Y lo que está en la memoria lo va a pasar al almacenamiento.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.



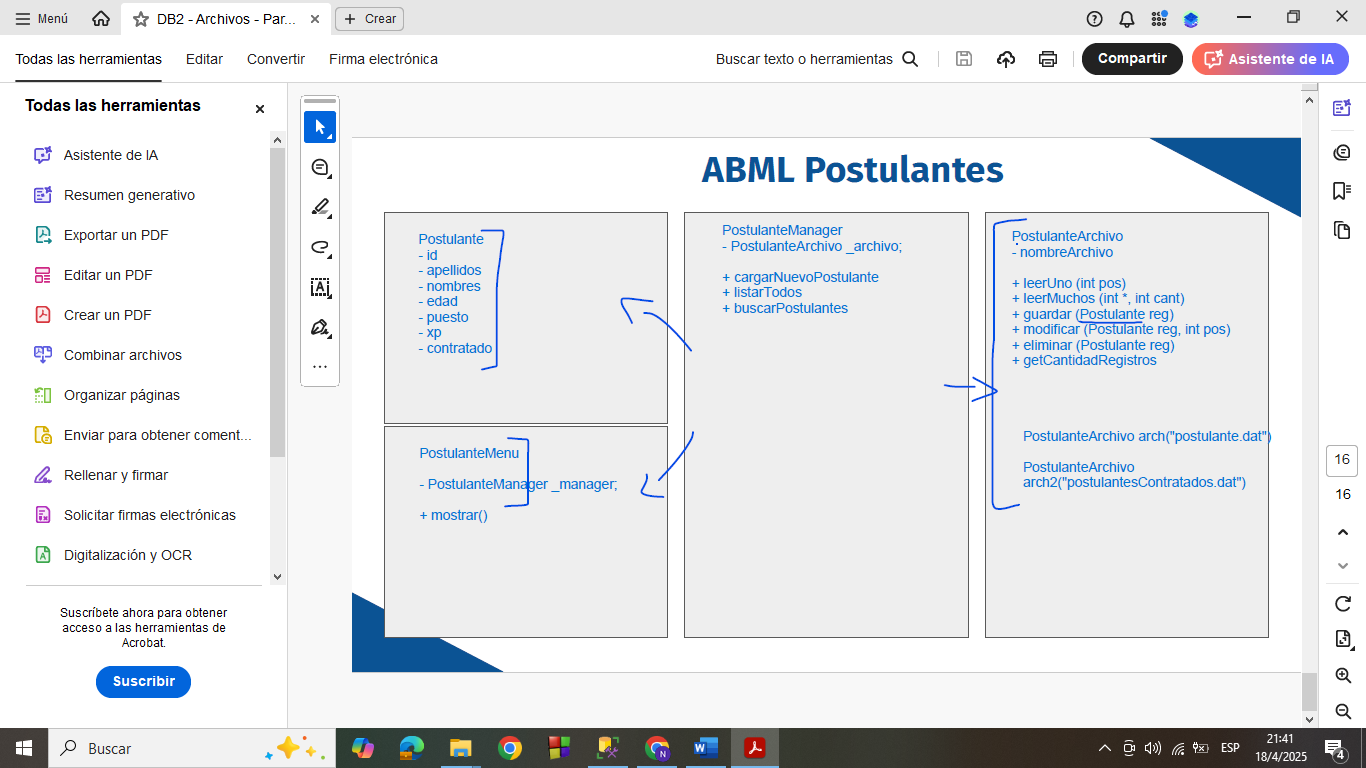
**fopen**: abre el archivo y devuelve un puntero file ya conectado para poder para poder trabajar. LOS parámetros que necesita son: el nombre del archivo y un modo de apertura que es un string

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

El archivo es un recurso escaso y lo podría llegar a necesitar otro programa u otro usuario, entonces lo usamos el tiempo que lo necesitamos y cuando no lo necesitamos más, lo cerramos.

**Ejemplo Practico**



**Cada una de las Clases va a tener una responsabilidad en particular:**

* **Postulante:** se encarga de crear un molde de un postulante, con las propiedades especificas que tiene que tenes un postulante y los métodos para construir ese postulante, para obtener el valor de las propiedades y para cambiar el valor de las propiedades
* **PostulanteManager:** se encarga de “manejar” todo lo relacionado a los postulantes: cargar a los postulantes, validar que los datos de entrada sean correctos, buscar a los postulantes, etc.
* **PostulanteArchivo** es la clase encargada de conectar la memoria heap con la stack a través de los archivos, sus métodos son guardar los registros, leer un registro, leer varios registros, eliminar un registro, modificar un registro**.** Como propiedad de clase va a tener el nombre del archivo, y lo podemos setear por defecto en el constructor
* **PostulanteMenu:** dibuja el menú de opciones y de acuerdo con la opción seleccionada, llama al método que corresponda.

**Pasos a Seguir Para Guardar un registro:**

**En la clase Postulante Archivo.cpp**

**bool PostulanteArchivo::guardar(Postulante registro)** 🡺 función que recibe como parámetro el objeto que se quiere guardar y retorna true o false, de acuerdo con si se guardó o no

1. Primero vamos a necesitar el puntero File:

**FILE \*pFile;**

1. Segundo vamos a necesitar abrir o crear el archivo con la función fopen:

**pFile = fopen(\_nombreArchivo.c\_str(), "ab");**

* Primer parámetro: nombre del archivo. String es de C++ y fopen es de C, por lo tanto, lo que va a recibir en texto va a ser un vector de char y como el nombre del archivo lo declaramos como string, ese string lo vamos a tener que convertir a char o mejor dicho enviarle el char del string y para ello utilizamos el c.str().
* Segundo parámetro modo de apertura: en este caso como lo que quiero es agregar los postulantes, voy a usar “ab” de appen. Si yo usara “wb”, cada vez que llamo a la función se me borra todo lo que ya escribí.

1. Comprobar si realmente pudimos abrir el archivo como escritura :

**if(pFile == nullptr){**

**return false;**

**}**

1. Función para escribir en el Archivo: fwrite()

Fwrite() devuelve la cantidad de registros que pudo escribir, entonces si la cantidad es cero

significa que no pudo escribirlo, por eso usamos un booleano para guardar lo que devuelve la función fwrite.

**Bool result,**

**result = fwrite(&registro, sizeof(Postulante), 1, pFile);**

* Recibe como primer argumento la dirección de memoria del objeto el registro que le pasamos por parámetro
* Segundo argumento el tamaño del registro ,en este caso del objeto, sería el sizeof de postulante
* Tercer argumento la cantidad la cantidad , estamos guardando uno
* Cuarto argumento el nombre del puntero file el puntero file el

1. Cerrar el Archivo con la función fclose() y retorna el resultado de la función fwrite()

**fclose(pFile);**

**return result;**

**En la clase PostulantesManager**

1. En la función CargarPostulante, creamos una estancia de postulante archivo con los valores por defectos que se establecieron en la clase PostulanteArchivo (va a recibir como nombre va a ser postulantes.dat)
2. Luego cargamos todos los datos del postulante
3. Creamos una Instancia de Postulante y le pasamos todos los datos que se pidieron
4. Creamos una estructura if y en la validación, llamamos a la función guardar del objeto PostulanteArchivo que se creó en el paso 6. De esta manera, verificamos si la función devolvió 1 o true, pudo guardar el postulante que creamos

**if(pArchivo.guardar(postulante)){**

**cout << "Se guardo correctamente!" << endl;**

**}**

**else {**

**cout << "Hubo un error inesperado, llame al de sistemas..." << endl;**

**}**

1. **Función para determinar cuántos registros hay cargados.**

* Es responsabilidad del postulante manager mostrar cuantos registros se cargaron

**void PostulanteManager::mostrarCantidadRegistros()**

**{**

**PostulanteArchivo pArchivo;**

**int cantidadRegistros = pArchivo.getCantidadRegistros();**

**cout << "La cantidad de registros son: " << cantidadRegistros << endl;**

**}**

* **Y es responsabilidad de Postulante Archivo, saber cuantos registros hay en el archivo**

Y para obtener la cantidad de registros, debemos dividir el tamaño total del archivo entre el tamaño un registro. Para ello vamos a utilizar dos funciones: **fseek() y ftell()**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Fseek() lo que hace es mueve el cursor adentro del archivo. Permite que a un a un puntero file nosotros le podemos decir cuántos bytes de desplazamiento nos queremos mover y desde donde (desde el principio, desde donde actualmente esto o desde el final del archivo)

Para ir al final básicamente lo que tendríamos que decir es que a nuestro puntero file se desplace 0 bytes desde el final y eso nos llevaría el cursor de donde está, dentro del archivo, hacia el final

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Ftell()nos dice cuántos bytes hay desde el principio del archivo hasta donde esté posicionado el cursor. Eso nos va a devolver entonces, si lo llevamos al final la cantidad de bytes totales del archivo.

Después esa cantidad total de bytes, la dividimos por el tamaño del registro sizeof(postulante) y obtenemos la cantidad de registro que tiene nuestro archivo.

En Postulantes Archivos:

**int PostulanteArchivo::getCantidadRegistros(){** *🡺 función para obtener la cantidad de registros*

int tamRegistro, total, cantidad;

**FILE\* pFile;** *🡺 función para obtener la cantidad de registros*

**pFile = fopen(\_nombreArchivo.c\_str(), "rb");** *🡺 abrimos el archivo con rb para lectura y para ubicar el puntero al principio del archivo*

**if(pFile == nullptr){***🡺 verificamos que se haya podido abrir*

**return 0;**

**}**

**// file byte desde**

**fseek(pFile, 0, SEEK\_END);** *🡺 movemos el cursos 0 bytes desde el final*

**total = ftell(pFile**);*🡺 ftell() no dice la cantidad de bytes que hay hasta donde está el cursos*

**cantidad = total / sizeof(Postulante);** *🡺 dividimos el tamaño del archivo entre el tamaño de 1 registro*

**fclose(pFile);** *🡺 siempre se cierra el archivo*

**return cantidad;***🡺 se retorna la cantidad de registros, para mostrarla en postulante manager*

**}**

1. **Listar todos los Postulantes:**

Como es una acción que se hace con los postulantes, la función de listar todos la vamos a escribir en el **Postulante Manager**. Pero la responsabilidad de abrir el archivo y leer los registros le corresponde al **Postulante Archivo.** Por eso en la clase **Postulante Manager** vamos a tener el siguiente método

**void PostulanteManager::listarTodos()**

**{**

**PostulanteArchivo pArchivo;** *🡺 creamos un objeto de la clase Postulante Manager*

**Postulante registro;** *🡺 creamos un objeto Postulante para guardar lo que me devuelve la función leer de postulante Archivo*

**int cantidadRegistros = pArchivo.getCantidadRegistros();** *🡺 obtenemos la cantidad de registros* ***para saber cuantas vueltas tiene que dar el for***

**for(int i=0; i<cantidadRegistros; i++)**

**{**

**registro = pArchivo.leer(i);** *🡺 por cada vuelta del for, llamo a la función que lee un Archivo y lo guardo en el objeto que cree (lo traigo a memoria)*

**cout << registro.toCSV() << endl;***🡺 y llamo a la función mostrar del objeto Postulante.*

**}**

**}**

En la clase **Postulante Archivo** vamos a tener el siguiente método:

**Postulante PostulanteArchivo::leer(int pos){** *🡺 función para leer 1 registro y recibe como parámetro la posición de registro que se quiere leer*

**FILE \*pFile;** *🡺 creamos el puntero*

**Postulante reg***;🡺 creamos un postulante o registro.*

**pFile = fopen(\_nombreArchivo.c\_str(), "rb");** *🡺 abrimos el archivo con rb, porque vamos a leer*

if(pFile == nullptr){ *🡺 verificamos si se pudo abrir*

**return reg;** *🡺 si no lo pudo abrir, devolvemos el postulante con el constructor vacío*

**}**

**fseek(pFile, sizeof(Postulante) \* pos, SEEK\_SET***);🡺 usamos esta función para mover el cursor la cantidad de posiciones que queremos (*sizeof(Postulante) \* pos) *para poder ir leyendo todos los registros, sino siempre leemos el primer registro.*

**fread(&reg, sizeof(Postulante), 1, pFile);** *🡺 con esta función leemos de a un registro*

**fclose(pFile*);****🡺 se cierra el archivo*

**return reg;** *🡺 se retorna el registro que se lee*

**}**