**Cuaderno de profesor**

**Introducción**

En este proyecto hemos realizado un cuaderno de profesor digital en el que hemos dividido los distintos módulos según el tipo de dato que fuese, pensamos que esta sería la mejor opción puesto que aportaría una mayor claridad y facilidad a la hora de la organización del grupo.

El programa te da la opción de gestionar los distintos tipos de datos, entre las cuales se encuentra crearlos, eliminarlos, modificarlos y mostrarlos. Cada tipo de dato hace eso y también vuelca a la estructura desde el fichero o viceversa.

En nuestro grupo cada integrante ha participado en la creación de dos módulos.

**Tecnología**

Hemos usado una serie de herramientas para la realización del programa:

* **Github:** Una plataforma en Internet en la cual subimos el código que vamos haciendo cada semana, también se puede ver el seguimiento de quién ha estado subiendo cada elemento.
* **Git:** Un programa necesario para poder realizar diversas operaciones en github, como subir algo, bajarlo, eliminarlo, etc.
* **Dev-c++:** Una herramienta en la que se ha programado el trabajo, nos sirve para poder escribir, compilar y ejecutar el programa.

**Módulos**

Diagrama

Descripción generada automáticamente

Los distintos colores son simplemente una decoración.

**Cristian Heredia Bravo** se encargó de:

* **Alumnos:** En este módulo realiza la gestión del tipo de dato alumno, pudiendo crear un nuevo alumno, eliminar cualquier alumno, modificar cualquier alumno, mostrar uno o todos los alumnos, encontrar un alumno dependiendo de su id, la cual es única y cargar alumnos desde el fichero a la estructura y viceversa.
* **Horarios:** En este módulo se realiza la gestión del tipo de dato horario, pudiendo crear un nuevo horario, eliminar cualquier horario, modificar cualquier horario, mostrar uno o todos los horarios, encontrar un horario pedido por el usuario y cargar horarios desde el fichero a la estructura y viceversa.

**Descripción de módulos**

**Alumnos**

Este módulo no tiene dependencia de otros módulos, debido a que es uno de los datos más básicos del programa.

El módulo posee una única estructura en la que se va a trabajar:

typedef struct {

char id\_alumno[7];

char nombre\_alum[21];

char direc\_alum[31];

char local\_alum[31];

char curso[31];

char grupo[11];

} alumno;

Esta estructura posee una serie de elementos propios de cada alumno, el id de cada alumno es único.

Ahora pasamos a las diversas funciones que posee el sistema:

---------------------------------------------------------------------------------------------------

//Cabecera: int encontrar\_alumno(alumno \*a, unsigned n, char \*)

//Precondición: vector de alumnos inicializado, la id tiene que ser de 7 carácteres

//Postcondición: Devuelve el número de elemento del alumno, devolverá

-1 si no existe

int encontrar\_alumno(alumno \*a, unsigned n, char \*cid);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

// Cabecera: void mostrar\_alumno (alumno)

// Precondición: alumno inicializado

// Postcondición: Muestra el contenido de alumno

void mostrar\_alumno (const alumno \*a);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: alumno \*leer\_alumnos(unsigned\*);

//Precondición: Ninguna

//Postcondición: Devuelve a través del puntero pasado el número de alumnos e inicializa el vector de alumnos

alumno \*leer\_alumnos(unsigned \*nAlumnos);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void guardar\_alumnos(const alumno \*alumnos, unsigned n)

//Precondición: Ninguna

//Postcondición: Escribe en el fichero Alumnos.txt los n alumnos pasados

void guardar\_alumnos(const alumno \*alumnos, unsigned n);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void menu\_admin\_alumno (alumno \*a, unsigned n)

//Precondición:Ninguna

//Postcondición: Muestra el menu de administrador para alumnos

void menu\_admin\_alumno();

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: int salir\_menu();

//Precondición: Ninguna

//Postcondición: Devuelve 1 si se quiere salir del menú y 0 si no

int salir\_menu();

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void agregar\_alumno(alumno \*\*a, unsigned \*n)

//Precondición: n es el número de alumnos y el vector a tiene que estar inicializado

//Postcondición: Agrega un nuevo alumno introducido por el usuario

void agregar\_alumno(alumno \*\*a, unsigned \*n);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void modificar\_alumno(alumno a)

//Precondición: Ninguna

//Postcondición: Modifica el alumno introducido según quiera el usuario

void modificar\_alumno(alumno \*a);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void mostrar\_alumnos (const alumno \*a, unsigned n)

//Precondición: n es el número de alumnos

//Postcondición: Muestra todos los alumnos del vector

void mostrar\_alumnos (const alumno \*a, unsigned n);

---------------------------------------------------------------------------------------------------

//Cabecera: void quitar\_salto(char \*s)

//Precondición: Ninguna

//Postcondición: Quita de la cadena el carácter '\n'

void quitar\_salto(char \*s);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void eliminar\_alumno(alumno \*\*a, unsigned n, int pos)

//Precondición: n es el número de alumnos y pos es la posición del alumno en el vector

//Postcondición: Elimina el alumno de la posición indicada

void eliminar\_alumno(alumno \*\*alumnos, unsigned \*n,char \*id);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: int existe alumno(alumno \*a, char \*s, unsigned n)

//Precondición: s es la id del alumno a encontrar y n es el número de alumnos

//Postcondición: Devuelve 1 si el alumno existe y 0 si no

int existe\_alumno (alumno \*a, char \*s, unsigned n);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void guardar\_alumno (const alumno \*a, FILE \*f)

//Precondición: Fichero abierto para escritura

//Postcondición: Guarda el alumno en el fichero

void guardar\_alumno(const alumno \*a, FILE \*f);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: int existe\_grupo(alumno \*a, unsigned n, char \*grupo)

//Precondición: Vector de alumnos a inicializado, n es el número de alumnos

//Poscondición: Devuelve 0 si no existe el grupo y 1 si existe

int existe\_grupo(alumno \*a, unsigned n, char \*grupo);

---------------------------------------------------------------------------------------------------

**Módulo horarios**

Horarios tiene dependencia con otros módulos así que usa algunas de sus funciones.

Este módulo usa estructuras de otros módulos, estas son la estructura de alumnos, materias y usuarios. Aunque tiene una estructura propia llamada horario:

typedef struct {

char id\_profesor[4];

int dia\_clase;

int hora\_clase;

char id\_materia[5];

char grupo[11];

} horario;

También posee dos funciones de ámbito privado:

---------------------------------------------------------------------------------------------------

//Cabecera: int grupo\_libre(const horario \*nuevo, const horario \*horarios, unsigned n)

//Precondición: n es el número de horarios, nuevo es el horario recibido que contiene el grupo hora y día que se quiere verificar si está libre

//Poscondición: Devuelve 1 si el grupo está libre a esa hora y día o 0 si no está libre

int grupo\_libre(const horario \*nuevo, const horario \*horarios, unsigned n);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: int profesor\_libre(const horario \*nuevo, const horario \*horarios, unsigned n)

//Precondición: n es el número de horarios, nuevo es el horario recibido que contiene el profesor (id), hora y día que se quiere verificar si está libre

//Poscondición: Devuelve 1 si el profesor está libre en esa fecha o 0 si no.

int profesor\_libre(const horario \*nuevo, const horario \*horarios, unsigned n);

---------------------------------------------------------------------------------------------------

Ahora pasamos a las funciones públicas:

---------------------------------------------------------------------------------------------------

//Cabecera: void menú\_admin\_horarios()

//Precondición: Ninguna

//Poscondición: Muestra el menú de administrador para horarios

void menu\_admin\_horarios();

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void agregar\_horario(horario \*\*hor, usuario \*usuarios, materia \*materias, alumno \*alumnos, unsigned nUsuarios, unsigned nMaterias, unsigned nAlumnos, unsigned \*nHorarios)

//Precondición: Todos los vectores deben estar inicializados, cada unsigned son el número de los distintos elementos de los vectores

//Poscondición: Agrega un horario teniendo en cuenta que exista los datos compartidos en otros ficheros

void agregar\_horario(horario \*\*hor, usuario \*usuarios, materia \*materias, alumno \*alumnos, unsigned nUsuarios, unsigned nMaterias, unsigned nAlumnos, unsigned \*nHorarios);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void eliminar\_horario(horario \*\*hor, int encontrado, unsigned \*nHorarios)

//Precondición: encontrado es la posición del horario que se desea eliminar y nHorarios es el número de horarios que hay

//Poscondición: Elimina un horario solicitado por el usuario.

void eliminar\_horario(horario \*\*hor, int encontrado, unsigned \*nHorarios);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void modificar\_horario(horario \*hor, materia \*materias, usuario \*usuarios, alumno \*alumnos, unsigned \*nHorarios, unsigned nMaterias, unsigned nUsuarios, unsigned nAlumnos, int encontrado);

//Precondición: Todos los vectores deben estar inicializados, cada unsigned son el número de los distintos elementos de los vectores, encontrado es el horario que se desea modificar.

//Poscondición: Modifica un horario solicitado por el usuario.

void modificar\_horario(horario \*hor, materia \*materias, usuario \*usuarios, alumno \*alumnos, unsigned \*nHorarios, unsigned nMaterias, unsigned nUsuarios, unsigned nAlumnos, int encontrado);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void mostrar\_horarios(horario \*horarios, unsigned nHorarios)

//Precondición: nHorarios es el número de horarios en el vector

//Poscondición: Muestra todos los horarios del vector

void mostrar\_horarios(horario \*horarios, unsigned nHorarios);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void guardar\_horarios(const horario \*horarios, unsigned n)

//Precondición: n es el numero de horarios y horarios es el vector en sí

//Poscondición: Guarda en el fichero todos los horarios

void guardar\_horarios(const horario \*horarios, unsigned n);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void guardar\_horario (const horario \*h, FILE \*f)

//Precondición: Fichero abierto para escritura

//Poscondición: Guarda en el fichero el horario recibido

void guardar\_horario (const horario \*h, FILE \*f);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: horario \*leer\_horarios(unsigned \*nHorarios)

//Precondición: Ninguna

//Poscondición: Inicializa el vector de horarios y almacena en nHorarios el número de horarios

horario \*leer\_horarios(unsigned \*nHorarios);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void mostrar\_horario(const horario \*hor)

//Precondición: Ninguna

//Poscondición: Muestra por pantalla el horario recibido

void mostrar\_horario(const horario \*hor);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void mostrar\_horarios(horario \*horarios, unsigned nHorarios)

//Precondición: nHorarios es el número de horarios que existen

//Poscondición: Muestra por pantalla el vector de horarios

void mostrar\_horarios(horario \*horarios, unsigned nHorarios)

---------------------------------------------------------------------------------------------------

Modulo Materias

El módulo de materias depende de por si mismo, no esta conectado con ninguna otra función, pero sí conectado a otros módulos por los cuales se puede llegar a el mismo.

Su estructura está compuesta por la Id de la materia (4 dígitos), el nombre de la materia(50 caracteres), y el nombre de la materia abreviada(3 caracteres).

typedef struct {

char Id\_materia [4];

char Nombre\_materia [50];

char Abrev\_materia [3];

} materia;

No posee ninguna función de ámbito privado, pero si posee unas 5 de ámbito público:

---------------------------------------------------------------------------------------------------

//Cabecera: void daraltasig(materia \*\*vMaterias,int \*nMaterias);

//Precondición: vMaterias es la posición de la materia que se desea dar de alta y nMaterias el número de materias que encontramos

//Postcondición: Da de alta la asignatura elegida

void daraltasig(materia \*\*vMaterias,int \*nMaterias);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void darbajaasig(materia \*\*vMaterias, int \*nMaterias);

//Precondición: vMaterias es la posición de la materia que se desea dar de baja y nMaterias el número de materias que encontramos

//Postcondición: Da de baja a la asignatura elegida

void darbajaasig(materia \*\*vMaterias, int \*nMaterias);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void modificarMateria(materia \*\*vMaterias,int nMaterias);

//Precondición: vMaterias es la posición de la materia que se desea modificar y nMaterias el número de materias que encontramos

//Postcondición: Se consigue modificar la materia exitosamente, dándole los valores correspondientes a los de la función “daraltasig”

void modificarMateria(materia \*\*vMaterias,int nMaterias);

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void listarMaterias(materia \*vMaterias,int nMaterias);

//Precondición: vMaterias es la posición de la materia que se desea modificar y nMaterias el número de materias que encontramos

//Postcondición: El vector, recorrerá todas las materias correspondientes mostrándolas por pantalla

void listarMaterias(materia \*vMaterias,int nMaterias);

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

//Cabecera: materia \*leer\_materia(unsigned \*nMaterias);

//Precondición: nMaterias es el número de materias que encontramos, cada unsigned son el numero distinto del vector

//Postcondición: Se encargará de leer todas las materias de nuestro fichero

materia \*leer\_materia(unsigned \*nMaterias);

---------------------------------------------------------------------------------------------------

Módulo Calificaciones

Este módulo tiene dependencia con otras funciones, así que se necesita la conexión de algunas de ellas con este módulo.

Como se puede apreciar en la estructura, esta depende de un carácter que sea la fecha de la calificación (8 dígitos), la descripción de la calificación (30 dígitos), la Id de la materia (4 dígitos, obtenida del módulo de materias), la Id del Alumno (6 dígitos, obtenida del módulo de Alumnos) y un entero que será el valor de la calificación.

typedef struct {

char Fecha\_calif [8];

char Descrip\_calif [30];

char Id\_materia [4];

char Id\_alum [6];

int Valor\_calif;

} calificacion;

Este módulo tiene cuatro funciones de ámbito público:

---------------------------------------------------------------------------------------------------

//Cabecera: void anadirCalif(char \*Id\_alum,calificacion \*\*vCalificaciones, int \*nCalificaciones,int \*nCalifAlumno);

//Precondición: Id\_alum será la id del propio alumno que deseamos encontrar, nCalificaciones el número de calificaciones que encontramos, nCalifAlumno será la calificación de dicho alumno y vCalificaciones la posición en la que nos encontramos del vector

//Postcondición: Anadir la calificación a un alumno correspondiente

void anadirCalif(char \*Id\_alum,calificacion \*\*vCalificaciones, int \*nCalificaciones,int \*nCalifAlumno);

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

//Cabecera: void listarCalif(char \*Id\_alum, calificacion\*vCalificaciones, int nCalificaciones, int nCalifAlumnos);

//Precondición: Id\_alum será la id del propio alumno que deseamos encontrar, nCalificaciones el número de calificaciones que encontramos, nCalifAlumno será la calificación de dicho alumno y vCalificaciones la posición en la que nos encontramos del vector

//Postcondición: Listar las calificaciones de un alumno seleccionado

void listarCalif(char \*Id\_alum, calificacion\*vCalificaciones, int nCalificaciones, int nCalifAlumnos);

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

//Cabecera:void eliminarCalif(char \*Id\_alum, calificacion \*\*vCalificaciones, int \*nCalificaciones, int \*nCalifAlumnos);

//Precondición: Id\_alum será la id del propio alumno que deseamos encontrar, nCalificaciones el número de calificaciones que encontramos, nCalifAlumno será la calificación de dicho alumno y vCalificaciones la posición en la que nos encontramos del vector

//Postcondición: Eliminar la calificación de un alumno

void eliminarCalif(char \*Id\_alum, calificacion \*\*vCalificaciones, int \*nCalificaciones, int \*nCalifAlumnos);

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

//Cabecera:void modifCalif(char \*Id\_alum, calificacion \*\*vCalificaciones,int nCalificaciones, int nCalifAlumnos);

//Precondición: Id\_alum será la id del propio alumno que deseamos encontrar, nCalificaciones el número de calificaciones que encontramos, nCalifAlumno será la calificación de dicho alumno y vCalificaciones la posición en la que nos encontramos del vector

//Postcondición: Modificar la calificación de un alumno seleccionado

void modifCalif(char \*Id\_alum, calificacion \*\*vCalificaciones,int nCalificaciones, int nCalifAlumnos);

**Alejandro Galafate Delgado** se encargó de:

* **Usuarios:** En este módulo se realiza la gestión del tipo de dato usuario. Contiene funciones que te permiten añadir un usuario nuevo, borrar uno ya existente, modificar uno de sus datos, ver la lista entera con todos los usuarios guardados en el fichero Usuarios.txt, puedes ver en detalle los datos de uno o todos los usuarios, buscar un usuario a traves de su Id, una función que te permite leer el fichero y volcarlo en la estrucura y por ultimo una función de menú para el administrador.
* **Profesor:** En este módulo se realiza la gestión del tipo de dato profesor. Contiene funciones que acceden al módulo calificaciones, horarios y alumnos para poder ver el horario que tiene el profesor que se ha logueado, permite ver una lista de los alumnos del grupo que el profesor desee y permite tanto cambiar calificaciones de un alumno, como poner una nueva o incluso borrar una.

**Usuarios**

Este módulo no tiene dependencia de otros módulos, debido a que es uno de los datos más básicos del programa.

El módulo posee una única estructura en la que se va a trabajar:

typedef struct {

char ID\_usuario[4];

char Nombre\_usuario[21];

char Perfil\_usuario[16];

char Usuario[6];

char Contrasena[9];

} usuario;

Esta estructura posee una serie de elementos propios de cada usuario, el id, el nombre, el usuaraio y la contraseña son propios de cada usuario.

Ahora pasamos a las diversas funciones que posee el sistema:

---------------------------------------------------------------------------------------------------

//Cabecera: usuario \*leer\_usuarios(unsigned \*nUsuarios);

//Precondición: el vector carga el unúmero de usuarios

//Postcondición: Devuelve a través del puntero pasado el número de usuarios e inicializa el vector de usuarios

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

// Cabecera: void ver\_estructura (const usuario \*user, unsigned n);

//Precondición: el vector carga la estructura y n como numero de usuarios

//Postcondición: Muestra todos los usuarios del vector cargado

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void ver\_usuario(const usuario \*user);

//Precondición: el veecor carga al usuario

// Postcondición: Muestra el contenido del usuario

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void ad\_usuario(usuario \*\*user, unsigned \*n);

//Precondición: n es el níumero de usuarios y el vector a tiene que estar inicializado

//Postcondición: Agrega el nuevo usuario que se introduce por teclado

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void guardar\_usuarios(const usuario \*usuarios, unsigned n);

//Precondición: Ninguna

//Postcondición: Escribe en el fichero Usuarios.txt los n usuarios introducidos por teclado.

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void borrar\_usuario (usuario \*\*usuarios,unsigned \*n,char \*id);

//Precondición: el vector carga la estructura, n es el numero de usuarios y el string carga el Id que intenta borrar el usuario.

//Poscondición: Elimina un usuario que selecciona el administrador.

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void edit\_usuario(usuario \*user);

//Precondición: el vecor carga la estructura

//Postcondición: Modifica al usuario introducido por teclado

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: int encontrar\_profesor(usuario \*usuarios, char \*idProfesor, unsigned nUsuario);

//Precondición: el vector ccarga la estructura, el string carga el Id del profesor y n el numero de usuarios

//Postcondición: Devuelve un -1 si la Id no coincide con la de un profesor y 0 si coincide

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: int existe\_usuario (usuario \*user, char \*s, unsigned n);

//Precondición: el vector carga la estructura, el string carga lo introducido por teclado y n es el numero de usuarios

//Postcondición: devuelve 1 si el Id está en uso y 0 si no lo está

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void cambio(char \*);

//Precondición: el string carga lo introducido por teclado

//Postcondición: Cambia los "\n" por un "\0"

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

//Cabecera: void menu\_para\_administrador\_de\_usuarios();

//Precondición: ninguna

//Postcondición: Añade, edita o borra los datos de usuario introucidos por teclado

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

**Profesor**

Este módulo tiene dependencia de los módulos usuaario, matrículas, Calificaciones y horarios.

El módulo no posee estructura en la que se vaya a trabajar.

Ahora pasamos a las diversas funciones que posee el sistema:

---------------------------------------------------------------------------------------------------

//Cabecera: void meniu\_profesor();

//Precondición: ninguna

//Postcondición: varía los datos de las calificaciones añadidas por teclado según el grupo y alumnoo seleccionado

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

**Adrián Jesús García** se encargó de:

* Menú Administrador: Este modulo se encarga de gestionar todas las acciones que se le permiten realizar a un administrador del sistema. Contiene funciones que acceden a los módulos usuario, alumnos, horarios y materias.
* Matricula: Este módulo tiene acceso a materias y alumnos y se encarga de gestionar las matriculas del fichero Matriculas.txt, permitiendo hacer acciones como dar de alta, dar de baja, modificar o listar matriculas y cargar matriculas desde el fichero a la estructura y viceversa.

**Menu Administrador**

Este modulo tiene acceso a los módulos usuario, alumnos, horarios y materias. No posee una estructura en la cual se vaya a trabajar. Solo posee una función, la cual es la siguiente:

---------------------------------------------------------------------------------------------------

//Cabecera: int menu\_administrador()

//Precondición: Ninguna

//Postcondición: Te permite acceder a los demás módulos según la elección que hagas

---------------------------------------------------------------------------------------------------

**Matricula:**

Este modulo depende de alumnos y de materias.

El módulo posee una única estructura en la que se va a trabajar:

typedef struct{

char Id\_materia [5];

char Id\_alumno [7];

}matricula;

Esta estructura posee un id de materia y un id de alumno, ambos deben coincidir con un id en sus respectivos ficheros (Materias.txt/Alumnos.txt)

Ahora pasamos a las diversas funciones que posee el sistema:

---------------------------------------------------------------------------------------------------

//Cabecera: void matriculas()

//Precondición: ninguna

//Postcondición: varía los datos de las matriculas añadidas por teclado según alumno seleccionado

---------------------------------------------------------------------------------------------------

//Cabecera: matricula \*leer\_matricula(unsigned \*nMatricula)

//Precondición: El vector carga el número de matriculas

//Postcondición: Devuelve a través del puntero pasado el número de matriculas

---------------------------------------------------------------------------------------------------

//Cabecera: int encontrar\_matricula(matricula \*b, unsigned m, char \*id)

//Precondición: el vector carga la estructura, el string carga el Id de la materia y m el número de matriculas

//Postcondición: Devuelve un -1 si la Id no coincide con la de un profesor y 0 si coincide

---------------------------------------------------------------------------------------------------

//Cabecera: void eliminar\_matricula(matricula \*\*matricula, unsigned \*m,char \*idmat)

//Precondición: el vector carga la estructura, m es el numero de matricula y el char carga el Id que intenta borrar el usuario.

//Postcondición: Elimina la matricula seleccionada

---------------------------------------------------------------------------------------------------

//Cabecera: void mostrar\_matricula (const matricula \*b)

//Precondición: ninguna

//Postcondición: muestra la matricula seleccionada

---------------------------------------------------------------------------------------------------

//Cabecera: void mostrar\_matriculas (const matricula \*b, unsigned m)

//Precondición: el vector carga la matricula y m es el numero de matriculas

//Postcondición: muestra todas la matriculas

---------------------------------------------------------------------------------------------------

//Cabecera: void agregar\_matricula(matricula \*\*b, unsigned \*n,char id)

//Precondición: el vector debe estar inicializado, n es el numero de matriculas y id es la materia a la cual le quieres inscribir

//Postcondición: agrega la matricula con los datos recibidos

---------------------------------------------------------------------------------------------------

//Cabecera: void guardar\_matriculas(const matricula \*matriculas, unsigned n)

//Precondición: ninguna

//Postcondición: Escribe en el fichero Matriculas.txt las n matriculas introducidos por teclado.

---------------------------------------------------------------------------------------------------

//Cabecera: void guardar\_matricula (const matricula \*b, FILE \*f)

//Precondición: ninguna

//Postcondición: Guarda la matricula en el fichero

---------------------------------------------------------------------------------------------------

**Pruebas de módulos**

En este apartado veremos algunas funciones de algunos de los módulos.

Empezamos con la función de agregar\_alumnos:

Texto

Descripción generada automáticamente

En este caso recibimos un puntero a un vector de alumnos y un puntero de unsigned (número de alumnos). El condicional se cumplirá si la función existe\_alumno devuelve un 0. Si se cumple el condicional se crea un nuevo alumno en la posición n del vector. La función de existe\_alumno es la siguiente:

Texto

Descripción generada automáticamente

En cada interacción del bucle se comprueba si la id pasada es la misma que la de cada id de cada alumno, en el caso que sea el mismo se devolverá un 1.

Ahora la función de eliminar\_horario:

Texto

Descripción generada automáticamente

En esta función se realiza un bucle que se repite desde encontrado hasta nHorarios-1, encontrado puede tener valores desde 0 hasta nHorarios-1, no se puede obtener otro valor porque en el vector no hay valores negativos ni superiores a nHorarios-1. El máximo valor que tendrá i será nHorarios -2, e i+1 tendrá el valor de nHorarios-1, el tamaño del vector es nHorarios y la última posición del vector es nHorarios-1, entonces en la última interacción será igualar el valor de la posición de nHorarios-2 a nHorarios -1.

Ahora la función darbajasig:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En esta función, tendremos que introducir el Id de la materia que querremos tratar, y para ello se encargara el vector nMaterias en recorrerse entero para encontrar si dicha materia se encuentra en el. Una vez encontrada la materia, se mandará un aviso que habrá que responder con un carácter, “s” para dar de baja dicha materia, eliminándose así dicha materia del fichero. Si no es así (Dando 0), devolverá que no hay materias existentes.

Ahora la función anadirCalif:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En esta función, seleccionado todos los valores para los punteros correspondientes, se encargara de añadir una calificación, modificando el fichero, añadiéndole una fecha, describiendo dicha calificación, introduciendo la Id de la correspondiente materia y alumno, y su calificación.

**Pruebas de integración**

Alumnos

Veamos ahora el funcionamiento del programa ejecutando esa función.

Texto

Descripción generada automáticamente

Primero probamos con una id ya existente y vemos que no nos deja continuar agregándolo, pero en el siguiente caso introducimos una id no existente y podemos continuar con la creación de un nuevo alumno.

Horarios

Imagen que contiene texto, tabla, escritorio, computer

Descripción generada automáticamente

En este caso solo hay un horario a eliminar, una vez se selecciona podemos observar que al listar horarios no aparece nada por eso mismo.