Proyecto de Metodología de la Programación 2022

CUADERNO DIGITAL

Jose Luis Venega Sánchez
Alejandro García Sánchez
Álvaro Periñán
Javier Rábago Montero

Índice

Documentación de los casos de prueba:

Descripción funcional del programa

La aplicación Cuaderno Digital es una implementación digital para llevar el control académico de todo el alumnado, como los datos del estudiante así como sus calificaciones, materias y sobre todo los grupos que el profesorado imparte junto con su horario. Pudiendo acceder, gestionar, crear, eliminar y modificar los datos de los alumnos o del propio profesor.

También cuenta con una administración la cual realiza tareas de administración con el fin de que no se produzca ningún error a la hora de la utilización de la aplicación.

En el Cuaderno Digital encontramos 2 tipos de Usuarios: Administradores y Profesor.

Si te registras como **Administrador** podrás realizar tareas de configuración de la aplicación, así como modificar el número de materias por defecto que puede cursar un alumno, el número de matrículas de alumnos en las materias, también podrás realizar tareas de gestión de alumnos, materias, usuarios (ya sean profesores u otros administradores).

Si te registras como **Profesor** podrás acceder a los datos de los alumnos a los cuales imparten clases, crear, eliminar o modificar un horario, acceder o añadir calificaciones.

IMPORTANTE: para un mejor uso del programa se tendrá que escribir en los ficheros de texto los datos necesarios para empezar a usar el programa.

Descripción del proceso de instalación

Para la instalación del programa será necesario tener la carpeta principal que contiene el archivo ejecutable y todos sus módulos implementados estarán contenidos en la carpeta Files, la cual a su vez está dividida en varias carpetas, siendo cada una de estas un módulo del programa principal .
Las carpetas son:

- **Configuración**: Contiene los ficheros de configuración (su módulo y correspondiente .txt) si no existe ninguno de estos la aplicación no podrá ser utilizada con éxito ya que almacena la configuración que se usará al iniciar la aplicación.

- **Usuario:** Contiene el módulo y su correspondiente .txt que almacena los usuarios que ya se han registrado en el sistema junto con sus IDs, nombre, perfil de usuario (Administrador o Profesor), nombre y sobre todo la contraseña.

- **Alumnos:** Contiene el módulo y su correspondiente .txt el cual almacena la información de los alumnos del centro que existirán al iniciar la aplicación, con sus id, nombre, dirección, localidad, curso y grupo al que pertenece.

- **Materias:**Contiene el módulo y su correspondiente .txt que almacena la información de las asignaturas que se imparten en el centro educativo junto con el ID, nombre de la materia y su abreviatura.

- **Matrículas:** Contiene el módulo y el correspondiente .txt que almacena la información sobre las asignaturas ya matriculados los alumnos una vez se inicie la aplicación, junto con el ID se encuentra un ID escolar.

- **Calificaciones:** Contiene el módulo y el correspondiente .txt que almacena la información sobre todas las calificaciones que obtengan los alumnos en las diferentes alumnos junto con la fecha de la calificación, una descripción, un ID de materia, el ID escolar y sobre todo la calificación que han obtenido.

- **Horarios:** Contiene el módulo y el correspondiente .txt que almacena la información de las materias que imparte cada profesor por ello contiene el ID profesor, un día de clase, una hora, el ID materia y el grupo al que pertenece dicho horario.

Descripción del proceso de ejecución

Antes de iniciar el programa todos los ficheros en los que tenemos guardado toda la información que usaremos en el transcurso del proceso, plasmarán toda su información en sus correspondientes estructuras de datos para luego trabajarlas.

- Inicio de sesión:

Al iniciar el programa, aparece un mensaje de bienvenida y pide al usuario que introduzca su nombre y su contraseña. Pueden darse los siguientes casos:

- El usuario no mete bien su nombre o no está registrado: El sistema permite al usuario crear un nuevo usuario o cerrar el programa. El nuevo usuario será por defecto participante y no tendrá privilegios ni de cronista ni de administrador.
- El usuario mete bien su nombre pero no la contraseña: El sistema muestra un mensaje de error y termina la ejecución del programa.
- El usuario acierta metiendo su usuario y su contraseña: Dependiendo del tipo de perfil que sea muestra los siguientes menús:
    - Menú Profesor
    - Menú Administrador

- Menú Profesor:

Si al iniciar sesión el programa detecta que el usuario es profesor, desplegará un menú en el que tendrá que introducir en que día de la semana desea trabajar (enumerados del 1 al 5 haciendo referencia como Lunes primer día de la semana y Viernes como el último). Y según sea el día que indique el usuario, aparecerá un listado con todos los grupos con los que trabaja el día que ha seleccionado. Y a continuación nos deja elegir con qué grupo deseamos trabajar. Y a partir de aquí desplegará un menú con dos opciones:

- **Lista de alumnos:** el cuál nos proporcionará la posibilidad de listar todos los alumnos que corresponden al grupo que hemos seleccionado de la materia que imparte dicho profesor.
  Al seleccionar un alumno nos dará la opción de ver los datos del alumno (también de cambiarlos en caso de que así lo desee), o también de trabajar con las calificaciones del alumno en la materia en cuestión, ya sea para examinarlas, añadir una nueva o eliminar una calificación.
- **Cambiar de grupo**: nos permite volver a seleccionar un grupo de los que trabaja el profesor ese día.

En caso de que el profesor decida parar de trabajar con el sistema, tendrá que negar que quiera seguir usando el programa cuándo este se lo pregunte.

- Menú Administrador:

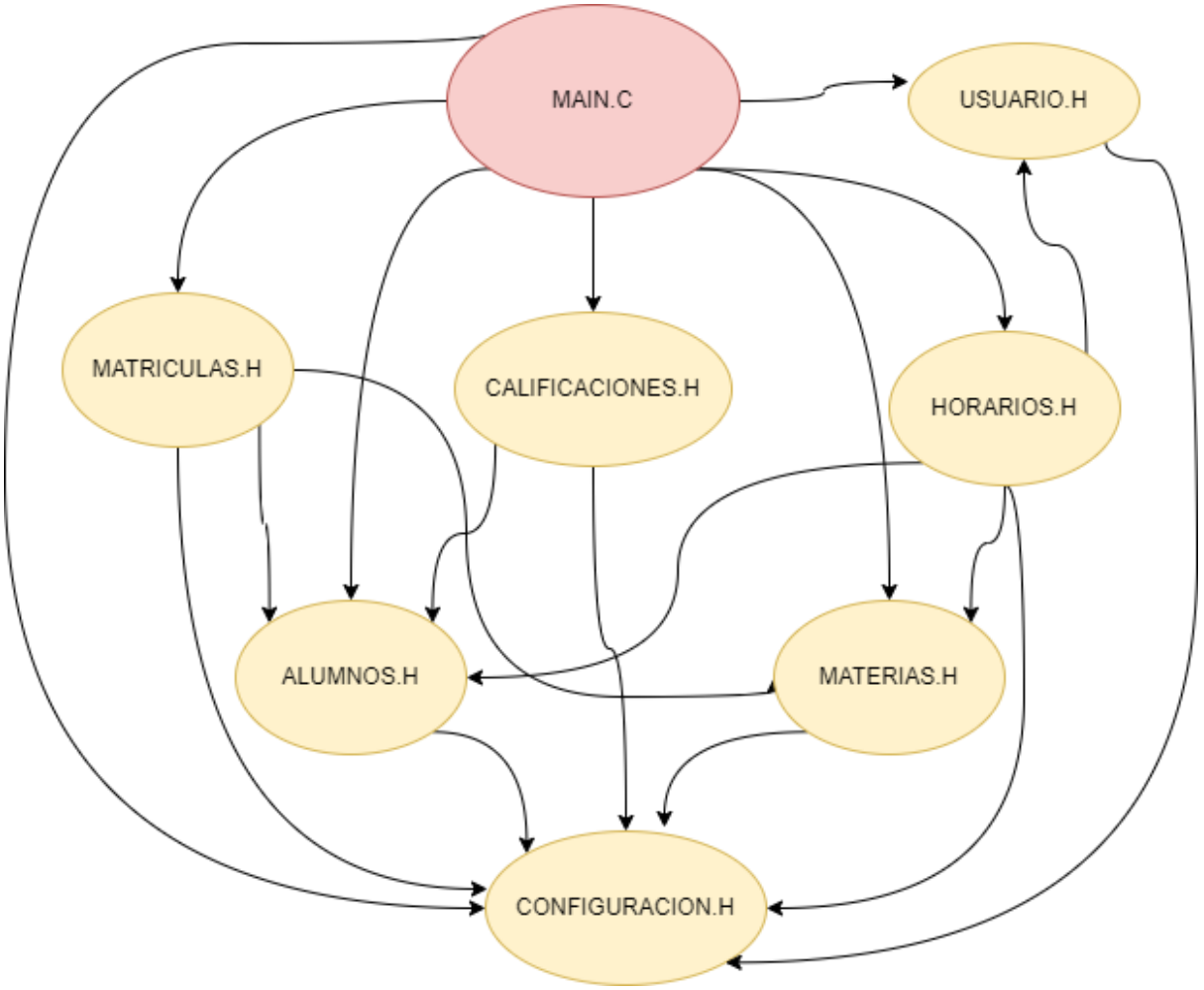
En caso de que el usuario que ha iniciado sesión es un administrador del sistema le aparecerá un menú con las siguientes opciones:

- **Usuarios:** Esta opción nos permite trabajar con el fichero de usuarios, es decir, con esta opción se nos desplegará otro submenú que nos habilita la posibilidad de crear, eliminar, modificar y listar los usuarios registrados en el sistema.

- **Alumnos:** Una opción similar a lo que hemos visto en el apartado de usuarios pero esta vez con los alumnos que están registrados en el sistema. Además, podremos seleccionar a un alumno en concreto y podremos ver todas las materias en las que se encuentra matriculado y podremos crear, eliminar o incluso modificar una matrícula en caso de que el administrador lo deseara.
- **Materias:** Con esta opción habilitará al administrador usar el fichero de texto de materias para trabajar con él, ya sea para listar, crear, modificar o eliminar las materias con sus respectivos datos.
- **Horarios:** Al igual que las anteriores opciones, permitirá al administrador gestionar los horarios que tienen los profesores asignados, es decir, el usuario podrá crear, eliminar, modificar o listar los horarios que están registrados en el sistema.
- **Salir del menú:** Con esta opción, como el propio nombre indica salimos del menú de administrador. Y por tanto saldremos del programa.

Cuando acabemos de utilizar el sistema, independientemente del tipo de usuario que haya entrado, se guardarán todos los datos que se han ido utilizando y trabajando en el transcurso del programa. Una vez hecho esto el programa finalizará.

Descripción de los módulos



- Main.c: Algoritmo principal.

- void menuAdmin(Usuario usuarios, int* nUsuarios, alumnos** v_alumnos, int* n_alumnos, materias** v_materias, int* n_materias, Matricula** matriculas, int* nMatriculas, Horario** Horarios, int* nHorarios, Configuracion** Configuraciones, int* nConfiguraciones),** menú donde están todas las opciones que le pueden aparecer a un administrador.

- void menuAdminAlum(alumnos v_alumnos, int* n_alumnos, Matricula** matriculas, int* nMatriculas),** submenú del menú de administrador para trabajar con los alumnos.

- void menuMatricula(Matricula matriculas, int* nMatriculas),** submenú del submenú de AdminAlum, donde podemos trabajar con las matrículas de los alumnos.
- void menuAdminMaterias(materias v_materias, int* n_materias),** submenú del menú de administrador, capaz de operar con las diferentes funciones referente a las materias.
- void menuAdminHorarios(Horario Horarios, int* nHorarios),** submenú del menú de administrador, capaz de operar con las diferentes funciones referente a los horarios.
- void menuProfe(int id_prof, alumnos v_alumnos, int* n_alumnos, materias** v_materias, int* n_materias, Calificacion** calificaciones, int* nCalificaciones, Matricula** matriculas, int* nMatriculas, Horario** Horarios, int* nHorarios, Configuracion** Configuraciones, int* nConfiguraciones),** menú donde están todas las opciones que le pueden aparecer a un profesor
- void menuListaAlumnos(int id_mat, alumnos v_alumnos, int* n_alumnos, materias** v_materias, int* n_materias, Calificacion** calificaciones, int* nCalificaciones, Matricula** matriculas, int* nMatriculas, Horario** Horarios, int* nHorarios, Configuracion** Configuraciones, int* nConfiguraciones),** submenú del menú de profesor, el cuál nos permite trabajar con los alumnos de la materia que imparte el profesor.
- static void bienvenida(), función que imprime por pantalla la bienvenida al programa.
- static void despedida(), función que imprime por pantalla la despedida al programa.

- Configuración: En este módulo se llevarán a cabo todas las acciones que tengan relación con la configuración del programa. Contiene las siguientes funciones.

- Configuracion* getConfiguracion(int *arraySize), donde "arraySize" contendrá el número de Configuraciones que hay en el fichero de texto y devolverá un puntero de estructuras "Configuración" que apuntará a un vector de elementos "arraySize" con sus correspondientes atributos conseguidos a través del fichero.

- void setConfiguracion(Configuracion *data, int i), actualiza en el fichero los datos contenidos en "data".

- void menuAdminConfiguracion(Configuracion Configuraciones, int nConfiguraciones),** imprime por pantalla las opciones de configuración y pide al administrador que selecciones una.

- void listarConfiguracion (Configuracion *Configuraciones, int nConfiguraciones), imprime por pantalla la configuración actual.

- void imprimeConfiguracion(Configuracion *c, int i), imprime por pantalla una configuracion c.

- void editarConfiguracion(Configuracion Configuraciones, int nConfiguraciones),** imprime por pantalla las configuraciones actuales y pide al administrador que elija una configuración para modificarla.

- int valorConfig (Configuracion* c, int nC, char const *var), devuelve en un entero el valor que busques en concreto

- Alumnos: En este módulo se llevan a cabo todas las acciones relacionadas con los alumnos. Contiene las siguientes funciones:

- void cargar_alumnos (alumnos **, int *), carga la lista de alumnos de un archivo de texto a una estructura del tipo “alumnos”.

- void guardar_alumnos (alumnos *, int), guarda todos los cambios realizados a la estructura el archivo de texto corresponiente.

- void crear_alumno(alumnos **, int *), añade a la estructura de alumnos una nueva entrada, con la condición de que el id introducido no existiese previamente.

- void eliminar_alumno(alumnos *, int *), imprime la lista de alumnos por pantalla y pide al usuario introducir la id del que desea eliminar.

- void eliminacion (alumnos **, int *, int), elimina un alumno de la estructura de alumnos, los reorganiza para no dejar ningún hueco y reduce el tamaño del vector de estructura.

- void listar_alumnos(alumnos *, int *), imprime por pantalla todos los alumnos.

- void mod_alumno(alumnos *, int *), pide al usuario el id del usuario a modificar, y luego el resto de datos a cambiar.

- void datos_alumno(alumnos *, int *, int), al recibir la id del alumno que queremos consultar imprime por pantalla todos sus datos.

- Materias: En este módulo se llevan a cabo todas las acciones relacionadas con las materias. Contiene las siguientes funciones:

- void cargar_materias (materias **, int *), carga la lista de alumnos de un archivo de texto a una estructura del tipo “materias”.

- void guardar_materias (materias *, int), guarda todos los cambios realizados a la estructura el archivo de texto corresponiente.

- void crear_materias(materias **, int *), añade a la estructura de materias una nueva entrada, con la condición de que el id introducido no existiese previamente.

- void eliminar_materias(materias **, int *), elimina una materia de la estructura de materias, las reorganiza para no dejar ningún hueco y reduce el tamaño del vector de estructura.

- void listar_materias(materias *, int *), imprime por pantalla todos las materias

.

- void mod_materia(materias *, int *), pide al usuario el id de la materia a modificar, y luego el resto de datos a cambiar.

- Matrículas: En este módulo se llevan a cabo todas las acciones relacionadas con las matrículas. Contiene las siguientes funciones:

- Matricula* getMatriculas (int *arraySize): Función que permite al usuario cargar en memoria las matriculas y les reserva un espacio en memoria.

- void setMatriculas(Matricula* data, int i): Función que permite al usuario guardar las matrículas en memoria y actualiza el fichero si hay algun cambio.

- void listarMatriculas(Matricula *data, int arraySize):Función que pone en una lista todas las matrículas.

- void imprimeMatricula(Matricula m): Función que permite al usuario imprimir por pantalla todas las matrículas

- void listaMateriasAlumno(Matricula *data, int arraySize, int id_alum): Función que permite al usuario ver todas las asignaturas que tienen los alumnos en la matrícula.

- void addMatricula(Matricula matriculas, int* nMatriculas, int id_alum):** Función que permite al usuario crear una nueva matrícula a un alumno ya existente o un alumno nuevo.

- void modificarMatricula(Matricula data, int nMatriculas, int id_alum):** Función que permite al usuario modificar las matrículas ya existentes en memoria.

- void eliminarMatricula(Matricula data, int* nMatriculas, int id_alum):** Función que permite al usuario eliminar cualquier matrícula guardada en memoria.

- void cambiarMateria(int id, Matricula data):** Función que permite al usuario cambiar las materias de cualquier matrícula.

- Calificaciones: En este módulo se llevan a cabo todas las acciones que estén relacionadas con las calificaciones. Contiene las siguientes funciones:

- Calificacion* getCalificaciones(int *arraySize), dónde “arraySize” contendrá el número de Calificaciones que hay en el fichero de texto y devolverá un puntero de estructuras “Calificacion” que apuntará a un vector de elementos “arraySize” con sus correspondientes atributos conseguidos a través del fichero.

- void setCalificaciones(Calificacion *data, int i), actualiza en el fichero los datos contenidos en “data”.

- void listarCalificaciones(Calificacion *data, int arraySize), muestra por pantalla la lista general de las calificaciones.

- void imprimeCalificacion(Calificacion c), imprime una calificación en específico.

- void listarCalificacionesAlumno(Calificacion *data, int arraySize, int id_alum, int id_mat), muestra por pantalla la lista de calificaciones de un alumno en una materia en concreto.

- void menuProfeCalificaciones(Calificacion calificaciones, int* nCalificaciones, int id_alum, int id_mat),** nos devuelve un menú que usaremos en el perfil de profesores para hacer las diferentes tareas que necesitan.

- void modificarCalificacion(Calificacion calificaciones, int nCalificaciones, int id_alum, int id_mat),** nos permite modificar una calificación de un alumno en una materia.

- void eliminarCalificacion(Calificacion calificaciones, int* nCalificaciones, int id_alum, int id_mat),** nos permite eliminar una calificación de un alumno en una materia..

- void addCalificacion(Calificacion calificaciones, int nCalificaciones, int id_alum, int id_mat),** nos permite crear una calificación de un alumno en una materia.

- void cambiarFecha(int id, Calificacion data),** nos permite cambiar la fecha de una calificación.

- void cambiarDesc(int id, Calificacion data),** nos permite cambiar la descripción de una calificación.

- void cambiarCalif(int id, Calificacion data),** nos permite la propia calificación.

- Horario: En este módulo se llevan a cabo todas las acciones que estén relacionadas con los Horarios. Contiene las siguientes funciones:

- Horario* getHorarios(int *arraySize), dónde "arraySize" contendrá el número de Horarios que hay en el fichero de texto y devolverá un puntero de estructuras "Horario" que apuntará a un vector de elementos "arraySize" con sus correspondientes atributos conseguidos a través del fichero.

- void setHorarios(Horario *data, int i), actualiza en el fichero los datos contenidos en "data".

- void listarhorario(Horario *Horarios, int nHorarios), muestra por pantalla la lista general de los horarios.

- void listarHorarioProfe(Horario *Horarios, int nHorarios, int id_prof), muestra por pantalla la lista de los horarios de dicho profesor.

- void imprimeHorario(Horario *h, int i), imprime los horarios.

- void modificarHorarioProfe(Horario Horarios, int nHorarios, int id_prof),** lista los horarios del profesor y pide que escriba el número del horario el cual se quiere modificar.

- void crearhorario(Horario Horarios, int* nHorarios,int id_prof),** pide que se cree un nuevo horario introduciendo todos los parametros necesarios (profesor, día, hora, materia, grupo).

- void eliminarHorario(Horario Horarios, int *nHorarios, int id_prof),** se pide que se seleccione un horario para que sea eliminado.

- void cambiarId(int id, Horario Horarios),**pide que se introduzca un id diferente para cambiar el id de un profesor.

- void cambiarDia(int id, Horario Horarios),** pide que se introduzca un día diferente para cambiar el día de un horario.

- void cambiarHora(int id, Horario Horarios),** pide que se introduzca una hora diferente para cambiar la hora ya puesta en un horario.

- void cambiarMat(int id, Horario Horarios),** pide que se introduzca una materia diferente para cambiar la materia ya puesta.

- void cambiarGrupo(int id, Horario Horarios),** pide que se introduzca un grupo de clase diferente para cambiar el que ya estaba puesto anteriormente

- Usuarios: En este módulo se llevan a cabo todas las acciones que estén relacionadas con los usuarios. Contiene las siguientes funciones:

- Horario* getUsuarios(int *arraySize), dónde "arraySize" contendrá el número de Usuarios que hay en el fichero de texto y devolverá un puntero de estructuras "Usuario" que apuntará a un vector de elementos "arraySize" con sus correspondientes atributos conseguidos a través del fichero.

- void setUsuarios(Usuario *data, int i), actualiza en el fichero los datos contenidos en "data".

- void menuAdminUsuarios(Usuario **usuarios, int* nUsuarios), devuelve el menú que corresponde a la opción que nos encontramos en el perfil de administrador, con este menú nos permitirá usar las diferentes funciones que desarrollamos en este módulo.

- void listarUsuarios(Usuario *data, int arraySize), muestra por pantalla la lista general de los usuarios.

- void imprimeUsuario(Usuario u), imprime un usuario determinado.

- void addUsuario(Usuario usuarios, int* nUsuarios),** nos permite crear un usuario.

- void cambiarUsuarioUsuario(int idVec, Usuario usuarios, int nUsuarios),** nos permite cambiar el nombre de un usuario.

- void cambiarPasswordUsuario(int idVec, Usuario usuarios, int nUsuarios),** nos permite cambiar la contraseña de un usuario.

- void cambiarNombreCompletoUsuario(Usuario usuarios, int nUsuarios),** nos permite cambiar el nombre completo de un usuario.

- void cambiarPerfilUsuario(int idVec, Usuario usuarios, int nUsuarios),** nos permite cambiar el perfil de un usuario.

- int existeNickUsuario(Usuario usuarios, int nUsuarios, char* nick, int indice),** nos devuelve 1 si existe el nick del usuario o 0 si no existe.

- int existeNombreUsuario(Usuario usuarios, int nUsuarios, char* usuario, int indice),** nos devuelve 1 si existe el nombre del usuario o 0 si no existe.

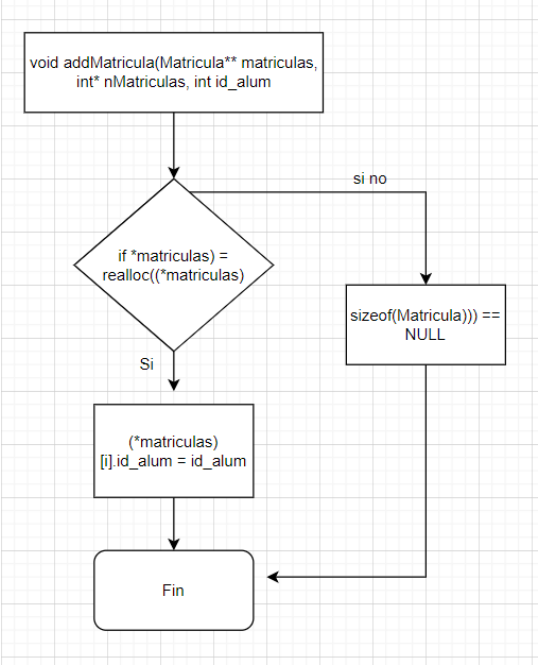
- void modificarUsuario(Usuario usuarios, int nUsuarios),** nos permite modificar un usuario.

- void eliminarUsuario(Usuario usuarios, int* nUsuarios),** nos permite eliminar un usuario.

- int login(int* nUsuario, Usuario usuarios, int* id),** con esta función nos devolverá 1 si el perfil del usuario es un administrador o 2 si el usuario es un profesor, en caso de que la contraseña es errónea, o el usuario decide no crear una cuenta devolverá -1, y en caso de que decida crearla entrará como profesor.

Documentación Casos de Pruebas

Void addMatricula(Matricula** matriculas, int* nMatriculas, int id_alum)



1.1 Complejidad ciclomática

V(G)=NPP+1=1+1=2

Otra forma de encontrar la complejidad ciclomática sería sumar el número de áreas encerradas que sería 1 más el área abierta que seria 1 también la suma de ambas daria 2

La primera entrada seria ver si hay espacio para crear la matrícula y si hay espacio se crea la matrícula juntandose el id alumno y el id materia.La salida de esta entrada seria la matricula creada por la combinación del id alumno y el id materia.La otra opuesta seria que no hubiera espacio en memoria para crear una nueva matrícula es este caso la función terminaría con un printf poniendo que no hay espacio suficiente espacio disponible y por lo tanto no habría ninguna salida porque el if no se cumple.

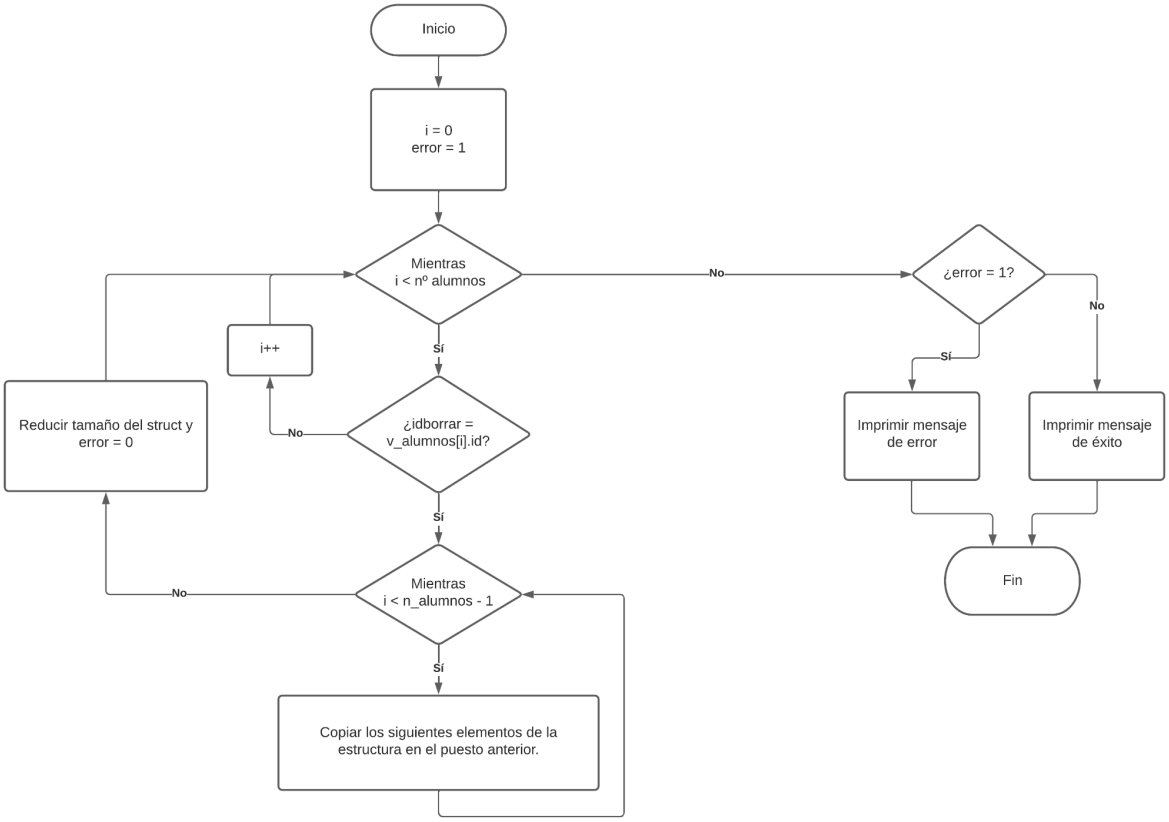
1.2 Prueba de caja negra

Para esta prueba aislamos la función a estudiar del resto del programa.

El único caso de error será cuando no haya espacio de memoria para crear una nueva matrícula.En este caso no se podrá crear ninguna matrícula nueva.En el resto, funciona perfectamente ya que crea una nueva matrícula.

Void eliminacion(alumnos **v_alumnos, int *n_alumnos, int idborrar)

2.1 Prueba de caja blanca



2.2 Complejidad ciclomática:

V(G) = NNP + 1 = 4 + 1 = 5

V(G) = NA - NN + 2 = 15 - 12 + 2 = 5

Otra manera de averiguar la complejidad ciclomática sería contar el número de regiones del grafo de control de flujo, donde claramente vemos cuatro, más el área que lo contiene todo cinco. Por tanto sabemos que esta función tendrá cinco posibles caminos diferentes, este número es relativamente bajo y por tanto con poco riesgo.

La primera ruta sería recibir el id de un alumno que esté presente en la estructura, se elimine correctamente poniendo el flag de error a 0 y recibiendo el mensaje de éxito. La ruta contraria sería recibir un id

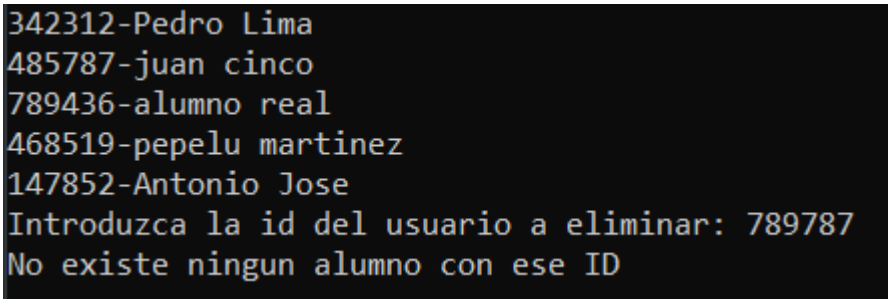
inexistente en la estructura, con lo que el flag de error seguiría a 1 y obtendríamos otro mensaje en consecuencia.

Las otras tres rutas restantes serían los bucles internos: buscar el id recibido en la estructura, copiar los datos que no queremos borrar en posiciones anteriores y reducir el tamaño del vector dinámico de la estructura.

2.3 Prueba de caja negra:

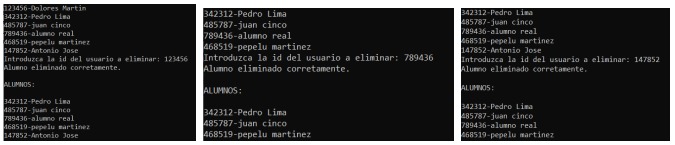
Para esta prueba aislaremos la función a estudiar del resto del programa. Tras cada valor enviado a la función se imprimirá por pantalla la estructura resultante, para comprobar el buen funcionamiento.

Empezaremos con los casos de error, en los que le damos a la función valores de todo tipo, ya sean enteros positivos que sabemos que no están en la estructura como negativos que no pueden estar. En todos los casos probados el resultado es satisfactorio, no se elimina ninguno de los datos de la estructura.



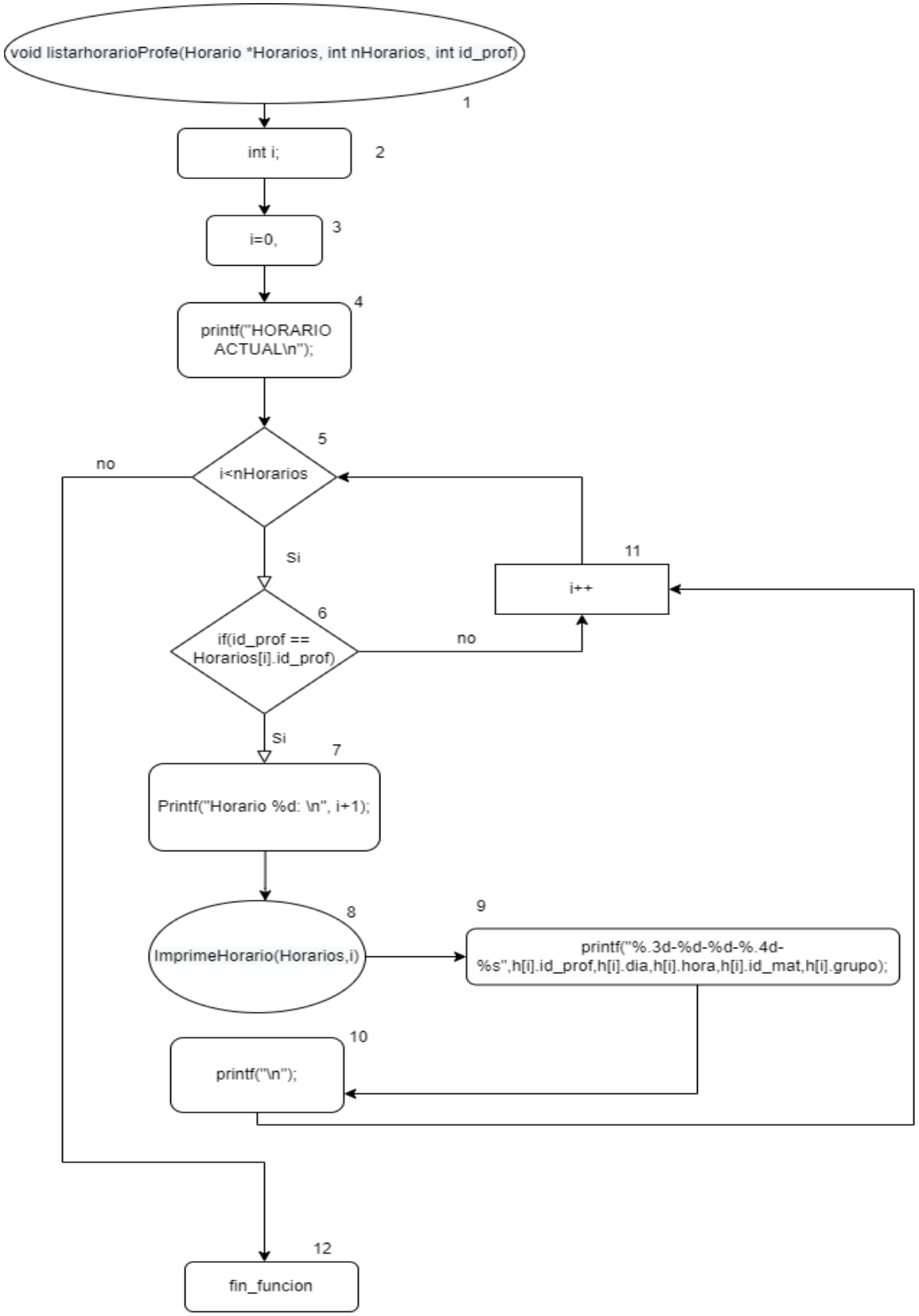
Para las pruebas de éxito de eliminar un alumno probaremos de varias maneras, tanto eliminando alumnos sin cerrar el programa como cerrándolo entre cada eliminación.

También tendremos en cuenta tanto eliminar alumnos que estén en los extremos de la lista como los que están en la mitad. Podemos observar que no vemos ningún error.



Void listarhorarioProfe(Horario *Horarios, int nHorarios, int id_prof)

3.1 Prueba de caja blanca



3.2 Complejidad Ciclomática

La complejidad ciclomática de una función cualquiera es:

V(G) = Nº Aristas - Nº Nodos + 2

V(G)= Nº Nodos Predicados + 1

Por tanto en esta función tenemos que la complejidad ciclomática:

V(G)= 13 - 12 + 2 = 3

V(G)= 2 + 1 = 3

La rutas posibles serían:

a) 1,2,3,4,5,6,7,8,9,10,11,12
b) 1,2,3,4,5,12
c) 1,2,3,4,5,6,11,12

3.3 Prueba de caja negra

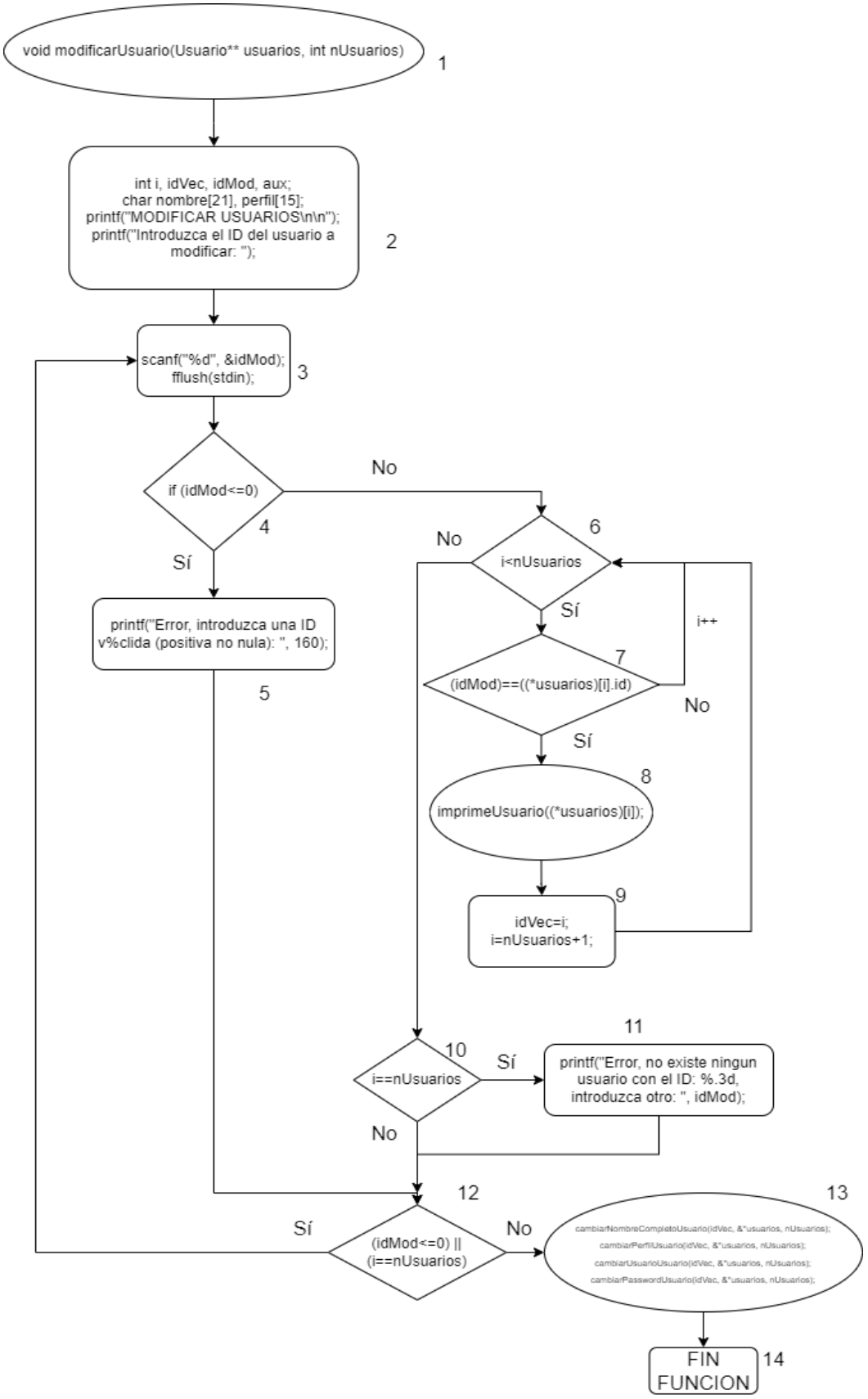
En este procedimiento declaramos 1 variable de tipo entero la cual es i, ya que solo tenemos un bucle, donde la variable i está inicializada a 0.

A continuación nos encontramos un bucle anidado a una función selectiva. Este bucle tiene por condición que el entero i ya declarado e inicializado a 0 debe de ser menor estricto que el tamaño del array horario (i < nHorarios). Si esto se cumple accedemos a una selectiva, pero si esto no se cumple saldremos del bucle y con ello de la función.

Si la condición del bucle es correcta, entramos en la selectiva donde comprueba si el id_prof tiene ya asignado un horario es decir (if (id_prof == Horarios[i].id_prof)). Si esto es correcto nos imprime el horario de dicho profesor con id_prof y aumentamos el contador del bucle volviendo al inicio del bucle (i ++). Sin embargo si no se satisface la condición de la selectiva volvemos al bucle hasta que encontremos un id_prof que tenga asociado un horario, sino, salimos del programa.

Void modificarUsuario(Usuario** usuarios, int nUsuarios)

4.1 Prueba de caja blanca:



4.2 Complejidad ciclomática:

Para calcular la complejidad ciclomática tenemos 3 formas:
V(G) = NA - NN + 2 = 16 - 12 + 2 = 6
V(G) = NNP + 1 = 5 + 1 = 6

También podemos averiguar la complejidad ciclomática contando el número de regiones que podemos ver en el grafo de control de flujo, y como podemos apreciar, podemos destacar 6 regiones. Por lo que, podemos intuir, que esta función podrá tomar 6 diferentes caminos dependiendo de los valores que nos encontremos.

Posibles caminos:

a) 1,2,3,4,5,12,3,4,6,7,8,9,6,10,11,12,13,14

b) 1,2,3,4,5,12,3,4,6,7,8,9,6,10,12,13,14

c) 1,2,3,4,6,7,8,9,6,10,12,13,14

d) 1,2,3,4,6,7,6,7,8,9,6,10,12,13,14

e) 1,2,3,4,6,7,8,9,6,10,12,13,14

f) 1,2,3,4,6,7,8,9,6,10,12,3,4,6,7,8,9,6,10,12,13,14

4.3 Prueba de caja negra:

Al entrar a la función se declaran las siguientes variables: i, idVec, idMod y aux (de tipo entero) y nombre y perfil (ambos strings de caracteres).

A continuación nos pide introducir un valor entero, en concreto idMod. Luego nos aparece una condición selectiva simple en la que nos verifica si la id que hemos puesto es positiva y no nula, en caso de que cumpla estas condiciones, nos dará un aviso de error.

En caso de que hayamos escrito bien la id, nos encontraremos con un bucle el cuál buscará por todo el vector de usuarios la id que queremos localizar y una vez la encuentre la imprimirá. En caso de que no haya encontrado la id en el vector, es decir, lo habrá recorrido entero y no ha dado resultado (i = nUsuarios) querrá decir que la id del usuario que hemos introducido no existe.

Todo lo explicado anteriormente se repetirá hasta que se encuentre un usuario con la id que hemos introducido ((idMod<=0) || (i==nUsuarios)).

Cuando finalmente hayamos encontrado al usuario con la id que hemos introducido nos ejecutará 4 funciones que nos permitirá cambiar cada uno de los aspectos del usuario exceptuando la misma id, obviamente.

Comentarios Adicionales

La distribución del proyecto se ha realizado de la siguiente forma:

Módulos de Configuración y Horario : Jose Luis Venega Sánchez.
Módulo de Matrículas: Álvaro Periñán.
Módulos de Alumnos y Materias: Javier Rábago Montero.
Módulos de Usuarios y Calificaciones: Alejandro García Sánchez.

El Main fue lo que se realizó en última instancia y fue realizado por Alejandro García Sáncehz y Jose Luis Venega Sánchez

Este enfoque solo nos orientó en las primeras semanas del desarrollo del programa, creando así las bases de la aplicación (implementación de los gets / sets y creación de las estructuras de datos. Cuando ya teníamos esa base general hecha procedimos a realizar las funciones específicas de cada módulo, basándonos en los dos perfiles que existen (Administrador y Profesor).

En el apartado de Prueba del Software, cada uno hemos realizado dicha prueba con una función o procedimiento que pertenece al módulo que ha desarrollado. Estas funciones o procedimiento son:

-void addMatricula(...) realizado por Álvaro Periñán
-void eliminacion(...) realizado por Javier Rábago Montero
-void listarhorarioProfe(...) realizado por Jose Luis Venega Sánchez
-void modificarUsuario(...)realizado por Alejandro García Sánchez