DOCUMENTACIÓN

INTEGRANTES: Alejandro Guerrero Medina, Mario Martínez Domínguez, Álvaro Santos Romero, Luis Fernando Torres Moya

DESCRIPCIÓN FUNCIONAL:

Se nos pide configurar un sistema basado en la liga de fútbol de primera división. Cimentado en editar plantillas de los futbolistas de dicha liga. Los usuarios pueden acceder mediante un sistema de LOGIN, pudiendo identificarse con tres roles:

–Participante.
–Cronista.
–Administrador.

Cada rol tendrá sus respectivas funciones a las cuales entraremos más tarde. Los cronistas puntúan a los jugadores, los administradores gestionan los datos de la liga y los participantes editan sus plantillas y juegan partidos con ellas. Finalmente habrá un ganador de la liga, que será el que tenga mayor puntuación.

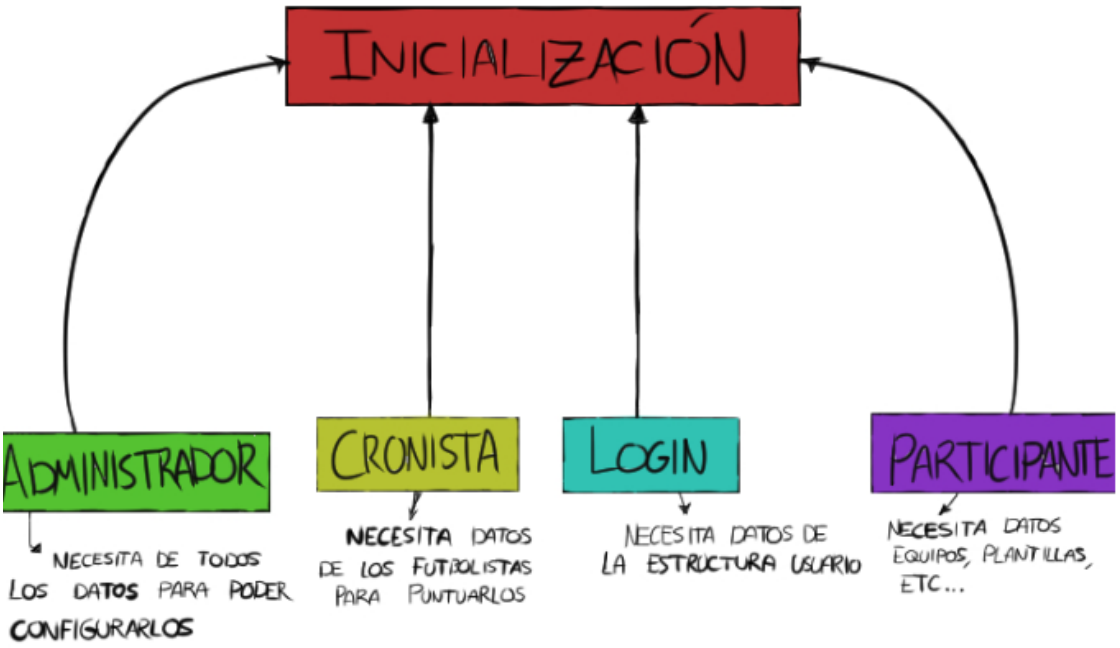
El sistema tendrá unas restricciones globales, que estarán guardadas en el fichero configuración:

–Tamaño ficheros.
–Tamaño de futbolistas por plantilla.
–Máximo de plantillas por participante.
–Presupuesto.
–Tamaño de los equipos actuales.
–Futbolistas actuales.
–Plantillas actuales.
–Usuarios actuales.
–Jugadores actuales en plantillas existentes.

En cuanto a los requisitos de cada rol: El participante puede crear, eliminar, listar y configurar plantillas, a la vez que consultar el ranking del sistema. En lo referente al cronista, sus actividades se reducen a listar y valorar a las distintas plantillas que hayan creado los usuario. Por último, el administrador tiene permitido configurar todos los aspectos, tanto de los equipos como de las plantillas que hayan creado los usuarios. También pueden modificar usuarios (borrarlos, por ejemplo) y modificar los propios valores de la configuración global.

DOCUMENTACIÓN DE DESCOMPOSICIÓN MODULAR:

Para la facilidad del problema, hemos decidido descomponer el problema en cuatro módulos: Inicialización, cronista, administrador, login y participante. Cada uno de estos se centra en las funciones que se espera que su rol ejecute. Veamos los módulos más a fondo:



–*Participante:* Como su nombre indica, el módulo participante contiene todas las funciones que puede realizar el usuario con rol de participante. Necesita a su vez que todos los datos de los ficheros estén ya en las estructuras. En su totalidad, el participante puede crear, modificar y listar todas sus plantillas. A su vez el participante también puede consultar el ranking de todas las plantillas que se encuentran en la liga.

–*Cronista:* En este módulo, se definen la función principal que va a tener este rol. La función de valorar equipos o plantillas. Como se ha dicho antes, después de cada jornada el cronista puede valorar a los equipos que existen en la liga. Haciendo uso de las estructuras que se han volcado desde los ficheros, se les da la nota. Como hemos dicho antes, hay un ranking. En este ranking se mirara la valoración total de todos los equipos y se listara.

–*Administrador:* El módulo administrador se encarga de proveer al usuario con rol de administrador de todas sus funciones. Modificar equipos existentes, modificar y eliminar usuarios o incluso modificar los datos que se encuentran en el fichero configuración. Este módulo es de los más importantes en ese sentido, pues los cambios que haga en el fichero configuración se aplican para todos los roles.

–*Inicialización:* Este módulo es el que se corresponde con el “core.h” del código. Este módulo se encarga de todos los volcados de los ficheros a las estructuras correspondientes.

Así mismo, también se encarga de ir actualizando los valores que se van cambiando en las estructuras durante la ejecución del programa en sus ficheros correspondientes. En este módulo encontramos tres tipos de funciones : Volcar, Mostrar y Escribir. Una para cada fichero que tenemos. Este módulo es el esqueleto de todo el problema.

–*Login:* Como indica el esquema anterior, el login también interactúa con el módulo de inicialización. Este módulo se encarga del acceso controlado de los usuarios al sistema. También contiene la función REGISTRO, la cual como indica su nombre, si un usuario no está registrado, se encarga de registrarlo en el sistema y de elegir la función que interpretará en la liga. Siendo esta de los tres roles que ya hemos especificado con anterioridad.

DOCUMENTACIÓN DE MÓDULOS:

1.-Cronista.h

Procedimiento menu_cronista:

Entran todas las estructuras que hemos volcado con anterioridad de los ficheros, esta función es una función de MENÚ. Es decir, esta función mostrará al usuario todas las opciones que tiene como rol Cronista. Tiene tres opciones, listar equipos actuales, valorar a los equipos actuales, y salir del programa finalmente. Listar ya forma parte de la librería del core.h, mientras que la valoración corre de cuenta del siguiente procedimiento.

Procedimiento valorar_futbolistas:

Recibe todas las estructuras. En este procedimiento, el usuario podrá valorar a todos los jugadores que desee, uno a uno. Se implementa usando un bucle que se ejecutará tantas veces como el usuario quiera valorar a equipos.

2.-Participante.h

Procedimiento menu_participantes:

Entran todas las estructuras que se han volcado anteriormente en el core.h desde los ficheros. Además, entra el id del participante. Como anteriormente, este es un procedimiento selectivo, se le enseña al participante todas las opciones que tiene con dicho rol. Usa una estructura selectiva simple.

Procedimiento crear_plantillas:

Entran todas las estructuras que se han volcado anteriormente en el core.h desde los ficheros. En este procedimiento, el participante podrá crear todas las plantillas que desee, acorde al presupuesto que tenga y al número de plantillas que ya posea (existe un máximo de plantillas como se dijo antes).

Procedimiento configurar_platillas:

El procedimiento recibe el ID del participante y todas las demás estructuras. Este procedimiento mostrará por pantalla un menú en el cual al usuario se le dan las diferentes opciones que tiene para configurar sus plantillas.

Procedimiento listar_jugadores_disponibles:

El procedimiento recibe el ID del participante y todas las demás estructuras. Mediante una serie de comparaciones, mostrará por pantalla una lista de todos los jugadores disponibles para fichar, junto con su ID y su Precio.

Procedimiento listar_jugadores_plantillas:

El procedimiento recibe la ID de la plantilla a listar, la estructura Jugadores Plantilla, la estructura Futbolistas y la estructura Configuración. Al igual que la anterior, esta hace una serie de comparaciones con la ID de plantilla para encontrarla en la estructura, y listar la por pantalla.

Procedimiento eliminar_plantillas:

El procedimiento recibe el id del participante y las estructuras Plantilla, Jugadores Plantilla y Configuración. Este pide una de las plantillas del participante a eliminar, la borra y reordena el vector dinámico de estructura plantillas restando un espacio.

Procedimiento Ranking:

El procedimiento recibe los vectores de Estructura Plantillas y Configuración. El procedimiento mostrará un ranking actualizado de las plantillas ordenadas por la puntuación más alta primero.

Procedimiento anadir_jugador_plantilla:

El procedimiento recibe la ID del participante, la plantilla a cambiar (identificada por una id) y todas las estructuras que se volcaron anteriormente. Lista los futbolistas que no tienen plantilla, de los cuales el participante elige al jugador que será añadido a su plantilla. Siempre y cuando se cumplan las condiciones indicadas, que tenga suficiente presupuesto para comprar al futbolista y que tenga hueco en la plantilla para dicho futbolista.

Procedimiento eliminar_jugador_plantilla:

El procedimiento recibe la ID del participante, la plantilla a cambiar (identificada por una id) y todas las estructuras que se volcaron anteriormente. La función muestra a los futbolistas de la plantilla que se recibe y posteriormente borra el futbolista seleccionado de la plantilla por el participante.

3.-Admin.h

Procedimiento menuAdministrador:

Entran todas las estructuras volcadas anteriormente en el core.h desde los ficheros. Se trata de un procedimiento selectivo; se le enseña al administrador las opciones de las que dispone mediante switches.

Procedimiento modificarEquipos:

Se reciben las estructuras equipos y configuración. Se le pide al administrador el nombre del equipo a modificar; en caso de que no coincida, se le requiere uno nuevo. En caso coincidente, este se mostrará por pantalla, junto con su id y su nombre, y a continuación se le asignará por teclado el nuevo nombre.

Procedimiento addEquipos:

Se reciben las estructuras equipos y configuración. Se le pide al administrador el nombre del equipo que va a crear; en caso de que exista uno con el mismo nombre, se le pide otro.

Procedimiento eliminarEquipos:

Se reciben las estructuras equipos y configuración. Se le pide al administrador el nombre del equipo que va a eliminar; en caso de que exista uno con el mismo nombre, se le pide otro.

Procedimiento addUsuarios:

Se reciben las estructuras usuarios y configuración. Se le pide al administrador el nombre del usuario que va a crear; en caso de que exista uno con el mismo nombre, se le pide otro.

Procedimiento modificarUsuarios:

Se reciben las estructuras usuarios y configuración. Se le pide al administrador el nombre del usuario que va a modificar; en caso de que exista uno con el mismo nombre, se le pide otro.

Procedimiento eliminarUsuarios:

Se reciben las estructuras usuarios y configuración. Se le pide al administrador el nombre del usuario que va a eliminar; en caso de que exista uno con el mismo nombre, se le pide otro.

Procedimiento modificarConfiguración:

Se recibe la estructura configuración. En este procedimiento, el administrador tiene la libertad de alterar la configuración completa de la misma.

4.-Login.h

Función acceso_sistema:

Se reciben las estructura usuario y configuración. Mediante esta función el usuario accede al sistema tras loguearse.

Procedimiento registro:

Se reciben las estructura usuario y configuración. Mediante este procedimiento se registra un usuario de tipo participante.

5.-Core.h

Procedimiento volcar_configuracion:

El procedimiento recibe el vector dinámico de estructura_config. Se vuelca toda la información del fichero configuracion.txt en el vector estructura estructura_config para así usarlo más sencillo en la ejecución del programa.

Procedimiento escribir_configuracion:

El procedimiento recibe el vector dinámico de estructura_configuracion. Se escriben en el fichero configuracion.txt los datos contenidos en la estructura_configuracion.

Procedimiento mostrar_configuracion:

El procedimiento recibe el vector dinámico de estructura_configuracion. Se muestran por pantalla los datos contenidos en estructura_configuracion.

Procedimiento volcar_futbolistas:

El procedimiento recibe el vector dinámico de estructura_config y estructura_futbolistas. Se vuelca toda la información del fichero futbolistas.txt en el vector estructura estructura_futbolistas para así usarlo más sencillo en la ejecución del programa.

Procedimiento escribir_futbolistas:

El procedimiento recibe los vectores dinámicos estructura_futbolistas y estructura_configuracion. Se escriben en el fichero futbolistas.txt los datos contenidos en la estructura estructura_futbolistas.

Procedimiento mostrar_futbolistas:

El procedimiento recibe los vectores dinámicos de estructura_configuracion y estructura_futbolistas. Se muestran por pantalla los datos contenidos en estructura_futbolistas.

Procedimiento volcar_equipos:

El procedimiento recibe el vector dinámico de estructura_config y la estructura_equipos. Se vuelca toda la información del fichero equipos.txt en el vector estructura estructura_equipos para así usarlo más sencillo en la ejecución del programa.

Procedimiento escribir_equipos:

El procedimiento recibe los vectores dinámicos estructura_equipos y estructura_configuracion. Se escriben en el fichero equipos.txt los datos contenidos en la estructura estructura_equipos.

Procedimiento mostrar_equipos:

El procedimiento recibe los vectores dinámicos de estructura_configuracion y estructura_equipos. Se muestran por pantalla los datos contenidos en estructura_equipos.

Procedimiento volcar_usuarios:

El procedimiento recibe los vectores dinámicos de estructura_config y estructura_usuario. Se vuelca toda la información del fichero usuario.txt en el vector estructura estructura_usuario para así usarlo más sencillo en la ejecución del programa.

Procedimiento escribir_usuarios:

El procedimiento recibe los vectores dinámicos la estructura_configuracion y estructura_usuarios. Se escriben en el fichero usuarios.txt los datos contenidos en la estructura estructura_usuarios.

Procedimiento mostrar_usuarios:

El procedimiento recibe los vectores dinámicos de estructura_configuracion y estructura_usuarios. Se muestran por pantalla los datos contenidos en estructura_usuarios.

Procedimiento volcar_plantillas:

El procedimiento recibe los vectores dinámicos de estructura_config y estructura_plantillas. Se vuelca toda la información del fichero plantillas.txt en el vector estructura estructura_plantillas para así usarlo más sencillo en la ejecución del programa.

Procedimiento escribir_plantillas:

El procedimiento recibe los vectores dinámicos de estructura_configuracion y estructura_plantillas. Se escriben en el fichero plantilla.txt los datos contenidos en la estructura estructura_plantillas.

Procedimiento mostrar_plantillas:

El procedimiento recibe los vectores dinámicos de estructura_configuracion y estructura_plantillas. Se muestran por pantalla los datos contenidos en estructura_plantillas.

Procedimiento volcar_jugadores_plantillas:

El procedimiento recibe los vectores dinámicos de estructura_config y estructura_jugadores_plantillas. Se vuelca toda la información del fichero jugadores_plantillas.txt en el vector estructura estructura_jugadores_plantillas para así usarlo más sencillo en la ejecución del programa.

Procedimiento escribir_jugadores_plantillas:

El procedimiento recibe los vectores dinámicos de estructura_configuracion y estructura_jugadores_plantillas. Se escriben en el fichero jugadores_plantillas.txt los datos contenidos en la estructura estructura_jugadores_plantillas.

Procedimiento mostrar_jugadores_plantillas:

El procedimiento recibe los vectores dinámicos de estructura_configuracion y estructura_jugadores_plantillas. Se muestran por pantalla los datos contenidos en estructura_jugadores_plantillas.

Procedimiento salir_programa:

Este procedimiento recibe todos los vectores dinámicos de tipo estructura existentes en la ejecución del programa. Este procedimiento nos sirve como una salida total de la ejecución del programa. Cuando es llamada, esta llama a todas las funciones de escritura en ficheros, actualizando así todos los datos de los ficheros por los que se han tenido en la última ejecución.

DESCRIPCIÓN DEL PROCESO DE INSTALACIÓN:

En lo referente a el proceso de instalación, tenemos dos pasos principales para poder utilizar nuestro programa como un ejecutable MakeFile (.mk).

Primero, debemos asegurarnos de tener instalada la herramienta CMake. Esta herramienta viene incluida en todos los sistemas operativos en base Linux. Si es el caso de que estamos en Windows, tenemos que estar seguros de instalarla antes de pasar íntegramente a la ejecución.

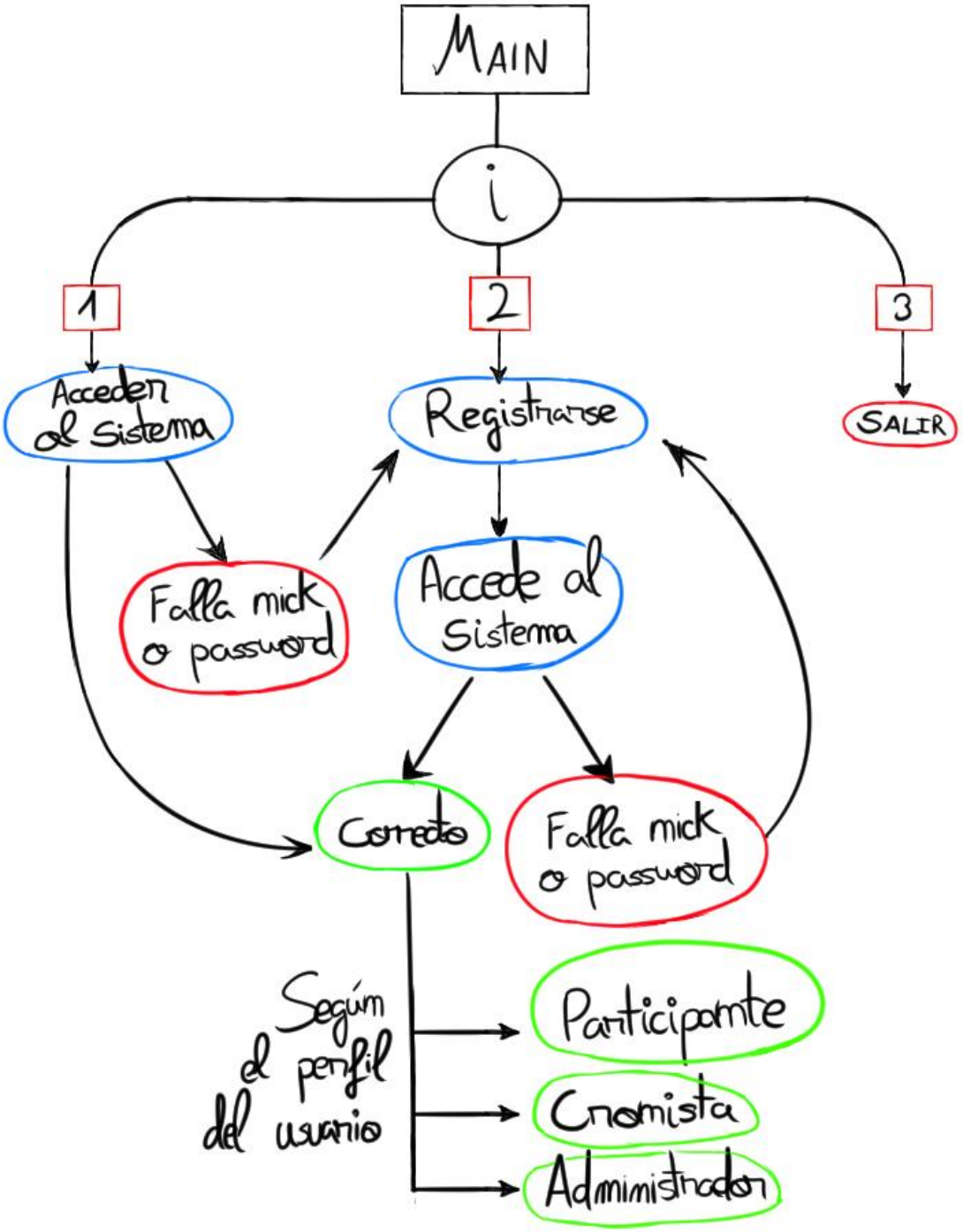
Pasemos a preparar la ejecución del .mk, tanto el windows como en Linux, habremos de poner el comando *make -f Makefile.mk main.* Con esto ya tenemos todo listo para simplemente ejecutar el MakeFile y poder usar el programa como un ejecutable.

DESCRIPCIÓN DEL PROCESO DE EJECUCIÓN DEL PROGRAMA:

Al ejecutarse la función main, en esta se declara, entre otras variables, el entero **«i».** Dicha variable es inicializada por el usuario tras sugerirle el programa que desea hacer: *acceder al sistema*, *registrarse* o *salir del programa*, sea cada caso asignado y referido a los tres primeros números naturales, respectivamente.

En caso de que el usuario quiera *acceder al sistema*, introducirá «1», asignándole a «i» dicho valor. Tras una comparación de igualdad mediante la estructura condicional «if», se llamará a una función del módulo *login.h* la cual comprobará internamente si el usuario está registrado o no. Si falla el nick o la contraseña, será redirigido a la función *registrarse*. Si el usuario asigna a la variable «i» el valor «2», comenzará el proceso de registro, una función alojada en el módulo login.h –como se ha comentado en el apartado anterior–. Por último, el valor «3» finalizará la ejecución del programa.

En caso de que se acceda correctamente al sistema –nick y contraseña sendos correctos–, se hará una comprobación mediante igualdades para que el sistema averigüe qué tipo de perfil es al que pertenece el usuario, redirigiendo en caso afirmativo al menú correspondiente –menú participante, menú cronista o menú administrador– y, finalmente, comience la ejecución del programa interno, donde entran en juego los diferentes módulos y funciones para llevar a cabo las tareas del programa en su conjunto.

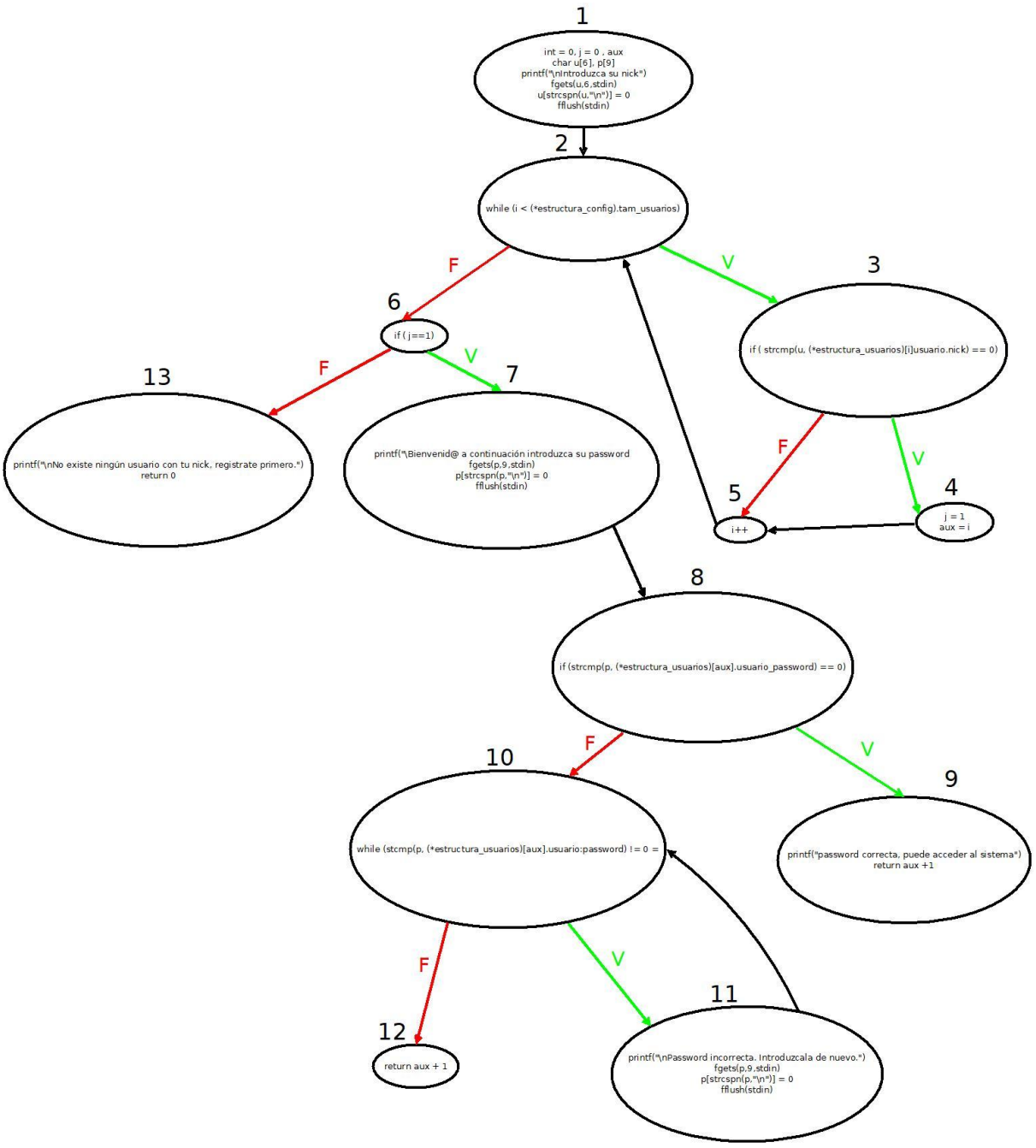

MAIN
i
1
2
3
Acceden al Sistema
Registrarse
SALIR
Falla nick o password
Accede al Sistema
Correcto
Falla nick o password
Según el perfil del usuario
Participante
Cronista
Administrador

DOCUMENTACIÓN DE LOS CASOS DE PRUEBA:

Función de Mario Martínez Domínguez: acceder sistema (login.h)

PRUEBAS DE CAJA BLANCA

1) GRAFO DE FLUJO



2) CÁLCULO DE LA COMPLEJIDAD CICLOMÁTICA

CC = Nº zonas delimitadas , las cuales son 4 (10-11 , 3-4-5 , 2-3-5 y el grafo exterior en sí)

Los caminos son los siguientes:

1. Nodos 1-2-6-13 Se da cuando no hay usuarios en el sistema registrados, por eso el nodo 2 es Falso.
2. Nodos 1-2-3-5-2-6-13 Cuando hay usuarios en el sistema pero el nick no es válido.
3. Nodos 1-2-3-4-5-2-6-7-8-9 Cuando hay usuarios en el sistema y se introduce correctamente el nick y contraseña.
4. Nodos 1-2-3-4-5-2-6-7-8-10-11-10-12 El caso en el que hay usuarios en el sistema , el nick se introduce correctamente pero la contraseña no, así que se le deja volver a introducirla hasta que la acierte.

PRUEBAS DE CAJA NEGRA

Creo casos de prueba a partir de darle valores válidos e inválidos a los parámetros de entrada y variables de la función, y añado el método AVL comprobando la funcionalidad de la función para los valores límites de correspondientes rangos. Estos parámetros son:

–La cadena “u”: tamaño seis que guarda el nick del usuario → vamos a darle un nick válido, un nick que sobrepase el tamaño y un nick de tamaño exactamente seis.

1. Nick válido. Ejemplo: “part1” –es el nick del único participante registrado en el sistema al comienzo de este–. Se consigue acceder al sistema. Devuelve “aux+1”, siendo “aux” la ID del usuario.
2. Nick que sobrepasa el tamaño. Ejemplo: “part1234”. Se consigue el resultado esperado ya que “fgets” recorta el tamaño máximo en caso de que se supere.
3. Nick de tamaño exactamente seis. Ejemplo: “part12”. El resultado obtenido es análogo al caso anterior.
4. Si seleccionamos un valor inválido de “u”. Ejemplo: “nick”, es decir, un valor no registrado, entonces la función devuelve cero.

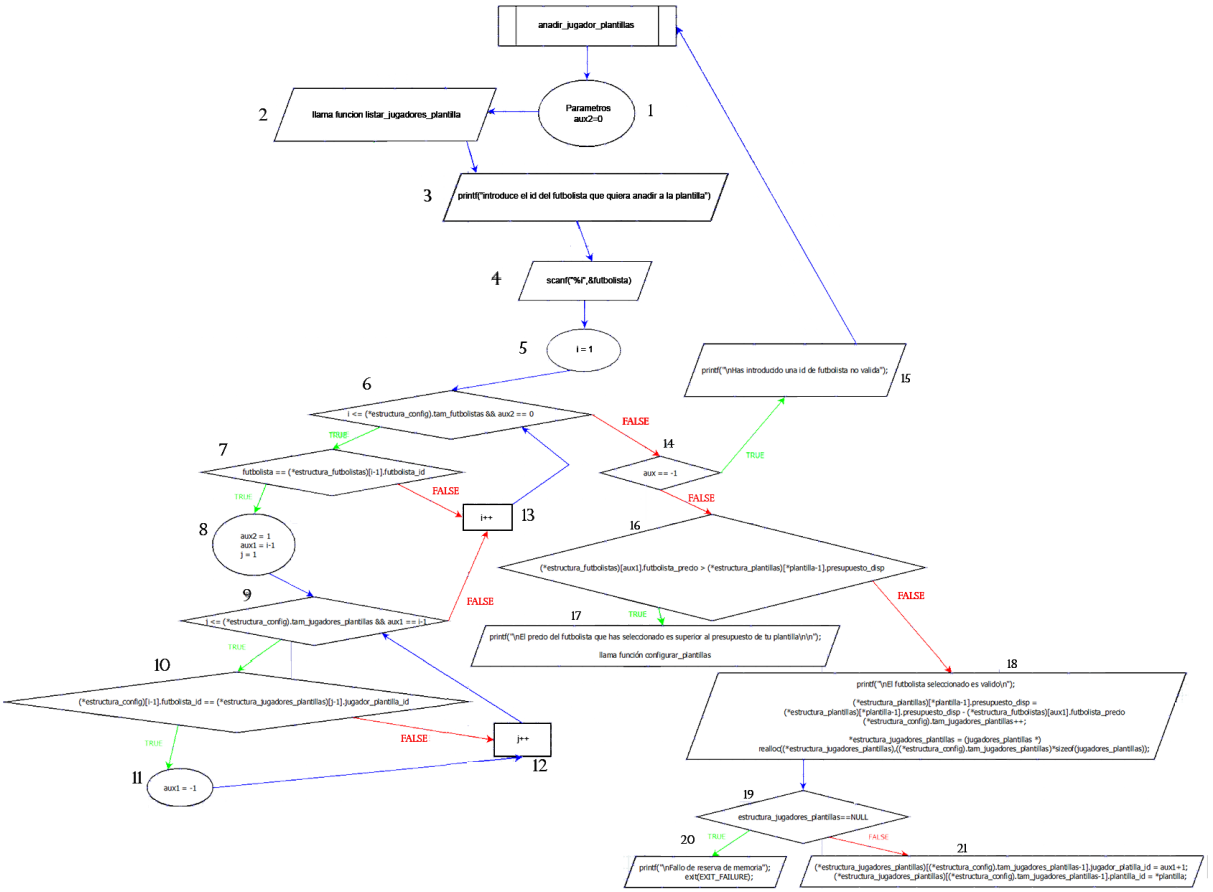
–La cadena “p”: tamaño nueve que guarda la contraseña del usuario → vamos a darle una contraseña válida, una contraseña que sobrepase el tamaño y una contraseña de tamaño exactamente nueve. Los resultados son análogos a los de la cadena “u” (nick).

–Si el parámetro de entrada *estructura_config –*estructura_config.tam_usuarios = 0–, es decir, no hay usuarios registrados en el sistema, entonces la función devuelve cero.

Función de Alejandro Guerrero Medina: anadir jugador plantillas (participante.h)

PRUEBAS DE CAJA BLANCA

1) GRAFO DE FLUJO



2) CÁLCULO DE LA COMPLEJIDAD CICLOMÁTICA

CC = Nº zonas delimitadas, las cuales son 6 (6-7-13, 6-7-8-9-13, 9-10-12, 10-11-12, 1-2-3-4-5-14-15, y el grafo exterior en sí)

Los caminos son los siguientes:

1. Nodos 1-2-3-4-5-6-14-15: se da cuando no hay futbolista.
2. Nodos 1-2-3-4-5-6-7-13-6-14-15: se da cuando no existe ningún futbolista con la ID introducida
3. Nodos 1-2-3-4-5-6-7-8-9-10-11-12-9-13-6-14-15: se da cuando la ID del futbolista introducida pertenece a otra plantilla.

4. Nodos 1-2-3-4-5-6-7-8-9-10-12-9-13-6-14-16-17: se da cuando el presupuesto de la plantilla es menor que el precio del futbolista.
5. Nodos 1-2-3-4-5-6-7-8-9-10-12-9-13-6-14-16-18-19-20: se da cuando hay fallo de reserva de memoria.
6. Nodos 1-2-3-4-5-6-7-8-9-10-12-9-13-6-14-16-18-19-21: se da cuando el futbolista deseado se añade con éxito a la plantilla seleccionada.

PRUEBAS DE CAJA NEGRA

Creo casos de prueba a partir de darle valores válidos e inválidos a los parámetros de entrada y variables de la función, y añado el método AVL comprobando la funcionalidad de la función para los valores límites de correspondientes rangos. Estos parámetros son:

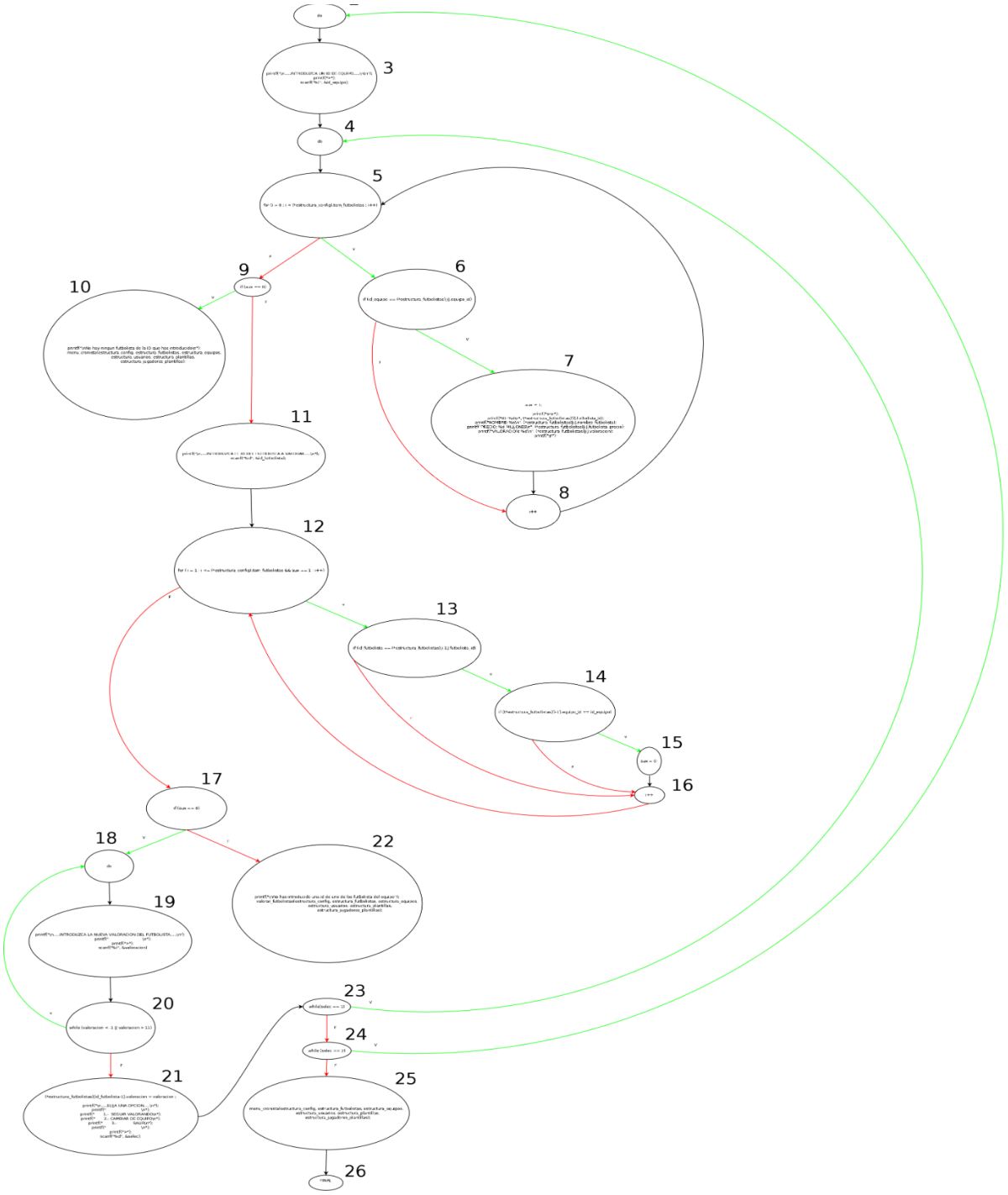
–Futbolista → tipo entero. Vamos a probar diferentes valores: valor válido (entre “1” y “*estructura_config.tam_futbolistas (número de futbolistas actuales)”, en este caso vamos a suponer que el máximo son 100. También probaremos un valor inválido (Ejemplo: valor decimal) y valores en los límites del rango válido: “0”, ”1”, “100”, “101”.

1. Ejemplo válido: “50”. Resultado esperado. Se añade el futbolista con ID “50” a la plantilla seleccionada exitosamente.
2. Ejemplos de valores que se encuentran en los límites del rango válido: para los valores “1” y “100” no da error, el resultado es el esperado. Para los valores “0” y “101” si hay error. Se llama a la función para que el participante vuelva a introducir una ID de un futbolista.
3. Ejemplo de valores inválidos: para un valor decimal. Ejemplo: “2,7” → se selecciona el futbolista con la parte entera de ese valor.

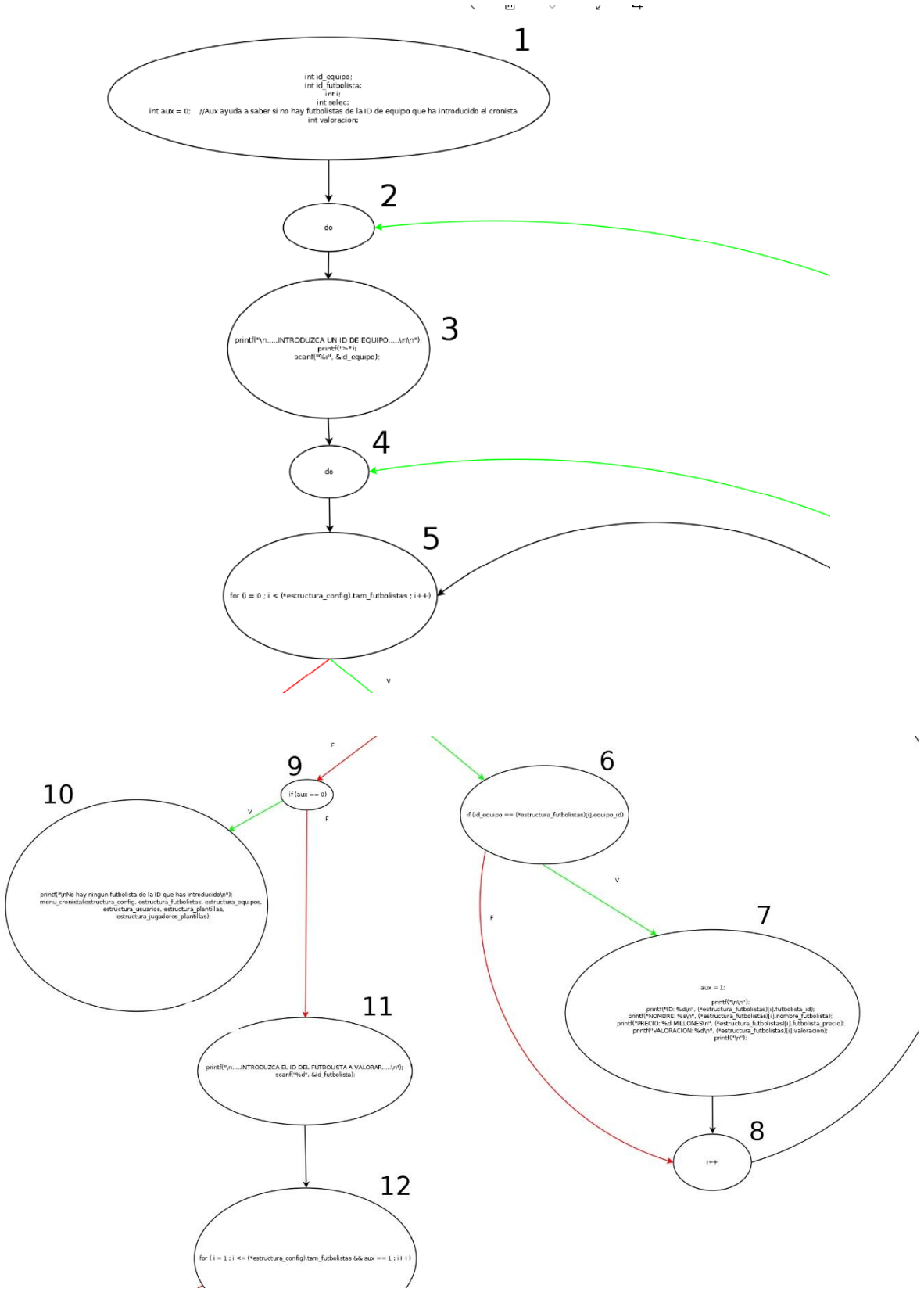
Función de Luis Fernando Torres Moya: Valorar Jugadores (cronista.h)

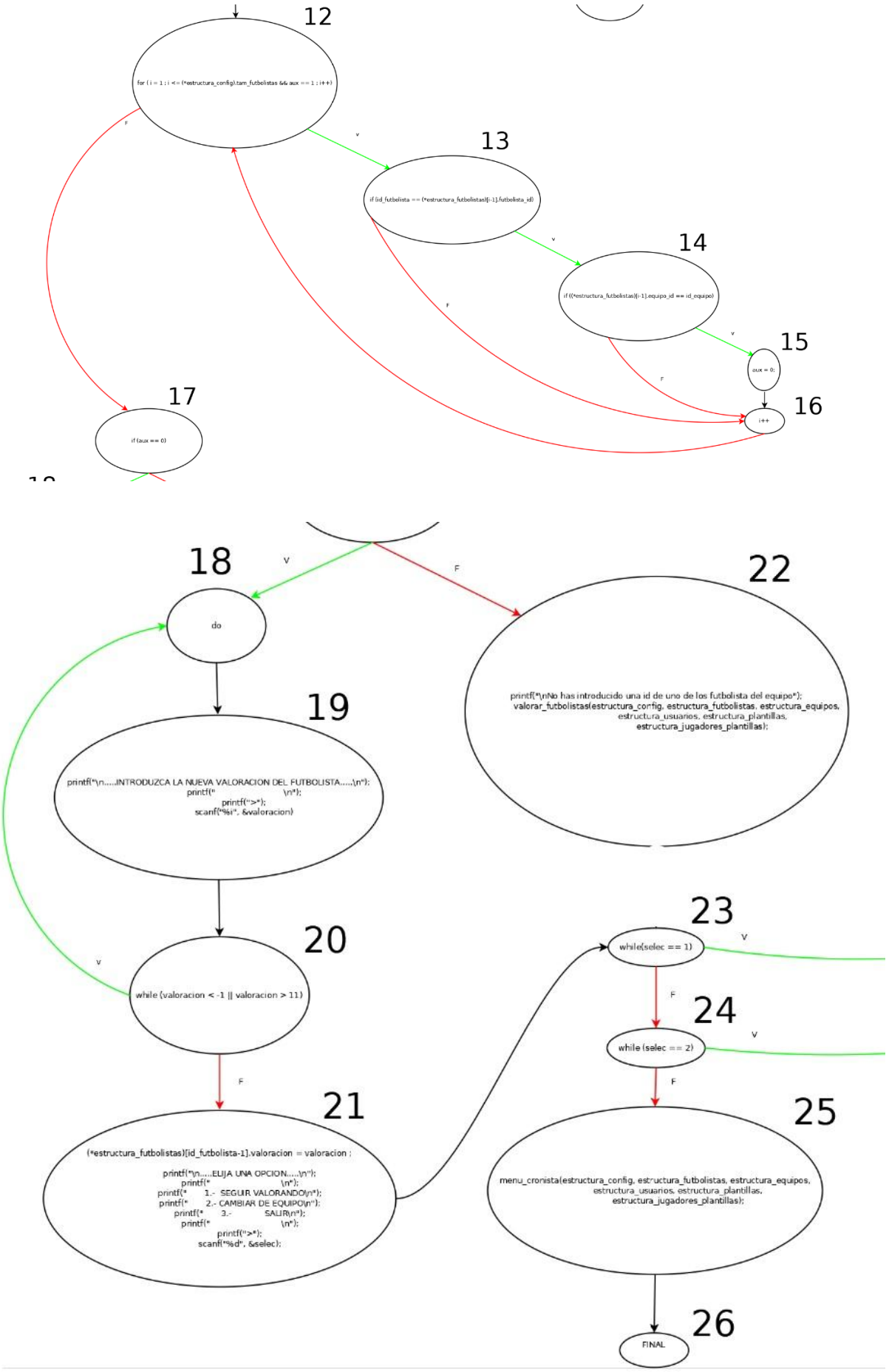
PRUEBAS DE CAJA BLANCA

1) GRAFO DE FLUJO



Veámoslo de una forma un poco más detallada:




12
for (i = 1 ; i <= (*estructura_config).tam_futbolistas && aux == 1 ; i++)
F
V
13
if (id_futbolista == (*estructura_futbolistas)[i-1].futbolista_id)
V
F
14
if ((*estructura_futbolistas)[i-1].equipo_id == id_equipo)
V
F
15
aux = 0;
16
i++
17
if (aux == 0)
18
V
F
do
19
printf("\n.....INTRODUZCA LA NUEVA VALORACION DEL FUTBOLISTA.....\n");
printf(" \n");
printf(">");
scanf("%i", &valoracion)
20
while (valoracion < -1 || valoracion > 11)
V
F
21
(*estructura_futbolistas)[id_futbolista-1].valoracion = valoracion ;
printf("\n.....ELIJA UNA OPCION.....\n");
printf(" \n");
printf(" 1.- SEGUIR VALORANDO\n");
printf(" 2.- CAMBIAR DE EQUIPO\n");
printf(" 3.- SALIR\n");
printf(" \n");
printf(">");
scanf("%d", &selec);
22
printf("\nNo has introducido una id de uno de los futbolista del equipo");
valorar_futbolistas(estructura_config, estructura_futbolistas, estructura_equipos,
estructura_usuarios, estructura_plantillas,
estructura_jugadores_plantillas);
23
while(selec == 1)
V
F
24
while (selec == 2)
V
F
25
menu_cronista(estructura_config, estructura_futbolistas, estructura_equipos,
estructura_usuarios, estructura_plantillas,
estructura_jugadores_plantillas);
26
FINAL

2) Cálculo de la complejidad ciclomática

CC = Nº zonas delimitadas, las cuales son 8

Veamos los caminos con más detalle:

1. 11-2-3-4-5-9-10: Se da cuando no existen futbolistas en la configuración.
2. 1-2-3-4-5-6-8-5-9-10: No existe ningún futbolista en el equipo de la ID proporcionada por el usuario.
3. 1-2-3-4-5-6-7-8-5-9-11-12-13-16-17-22: No existe ningún futbolista con la ID de futbolista proporcionada.
4. 1-2-3-4-5-6-7-8-5-9-11-12-13-14-16-12-17-22: Se ha introducido la ID de un futbolista que no está en el equipo seleccionado.
5. 1-2-3-4-5-6-7-8-5-9-11-12-13-14-15-16-12-17-18-19-20-21-22-23-24-25: Se elige la opción de salir de la función o cualquier otro valor que no sea 1 o 2.
6. 1-2-3-4-5-6-7-8-5-9-11-12-13-14-15-16-12-17-18-19-20-21-22-23-4: Se elige la opción de seguir valorando a los jugadores del equipo.
7. 1-2-3-4-5-6-7-8-5-9-11-12-13-14-15-16-12-17-18-19-20-21-22-23-24-2: Se elige la opción de cambiar de equipo para valorar a jugadores de otro equipo.
8. 1-2-3-4-5-6-7-8-5-9-11-12-13-14-15-16-12-17-18-19-20-21-22-23-24-2-3-4: Camino que se da cuando se repite el bucle do while y se quiere seguir valorando.

PRUEBAS DE CAJA NEGRA

Creo casos de prueba a partir de darle valores válidos e inválidos a los parámetros de entrada y variables de la función, y añado el método AVL comprobando la funcionalidad de la función para los valores límites de correspondientes rangos. Estos parámetros son:

-*equipo_id*: Vamos a darle tres valores, una ID que este bien, una ID que este mal y una ID este vacia de futbolistas. En un principio tenemos 20 equipos, numerados con ID de 01 a 20.

-*futbolista_id*: Le daremos 2 valores, una ID que se encuentre en las estructuras y otra ID de futbolista que no lo esté. Todos los valores son válidos, siempre y cuando se encuentren el el fichero de futbolistas. Principalmente tenemos 100 futbolistas, numerados del 00 al 99.

-*valoración*: A valoracion, se le daremos tres valores, un valor que esté bien, uno fuera de los valores admitidos y otro que esté justo al límite. La valoración correcta está entre 0 y 10, cualquier otra cosa devuelve error y pide introducirlo otra vez.

-*tam_futbolistas*: Al ser un valor que se pasa por parámetros, tenemos tres tipos de valor, uno aceptable, otro que de error y otro que esté en los límites. El tamaño de futbolistas es el que traiga de por si el fichero de configuración. Eso sí, si tenemos 0, veamos más adelante en los ejemplos lo que pasaría.

EJEMPLOS DE FUNCIONAMIENTO Y SU SALIDA

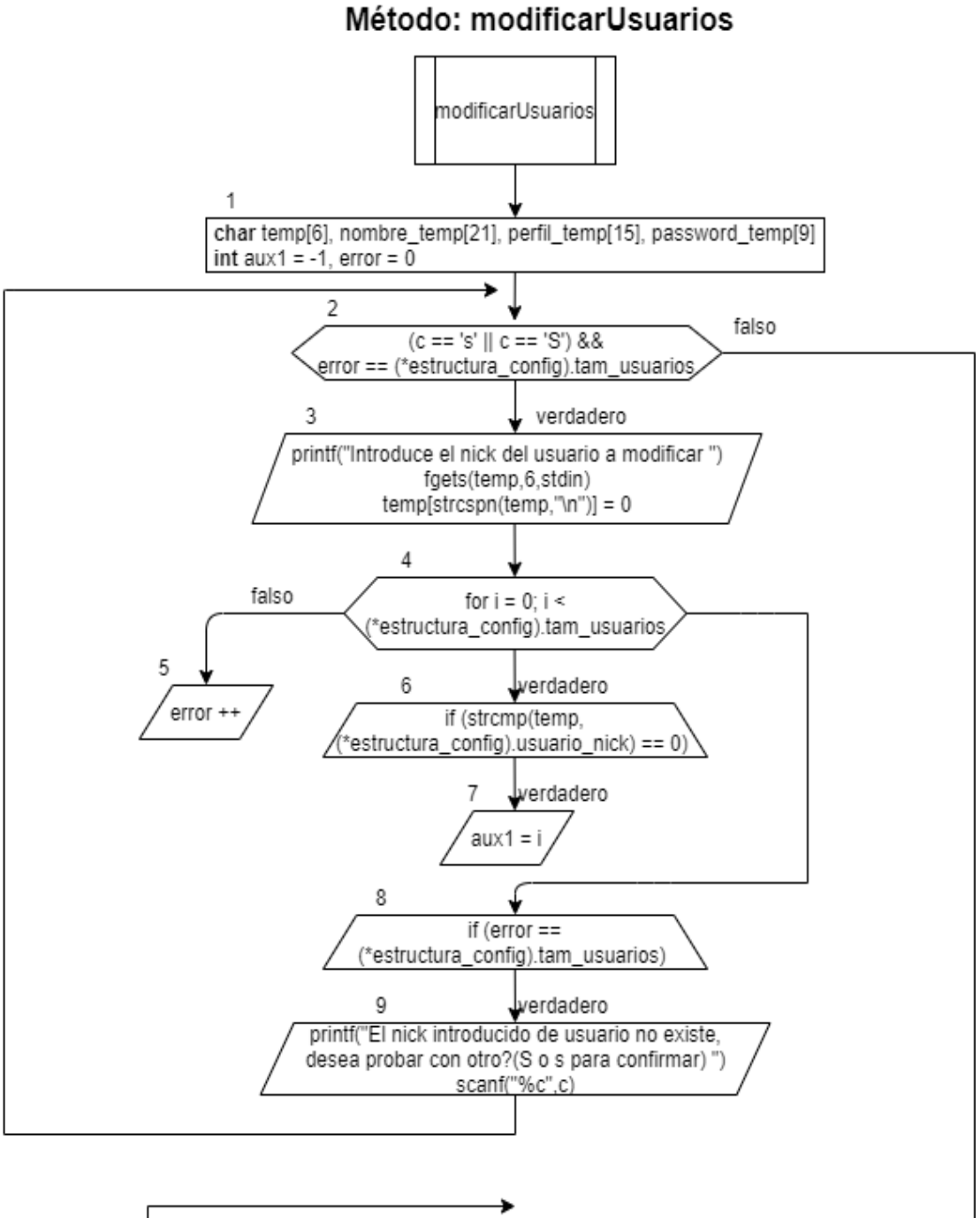
	Entrada		
	Valido	No Valido	Limite
equipo_id	05	35	00
futbolista_id	22	105	
valoracion	5	23	10'5
tam_futbolistas	200	0	

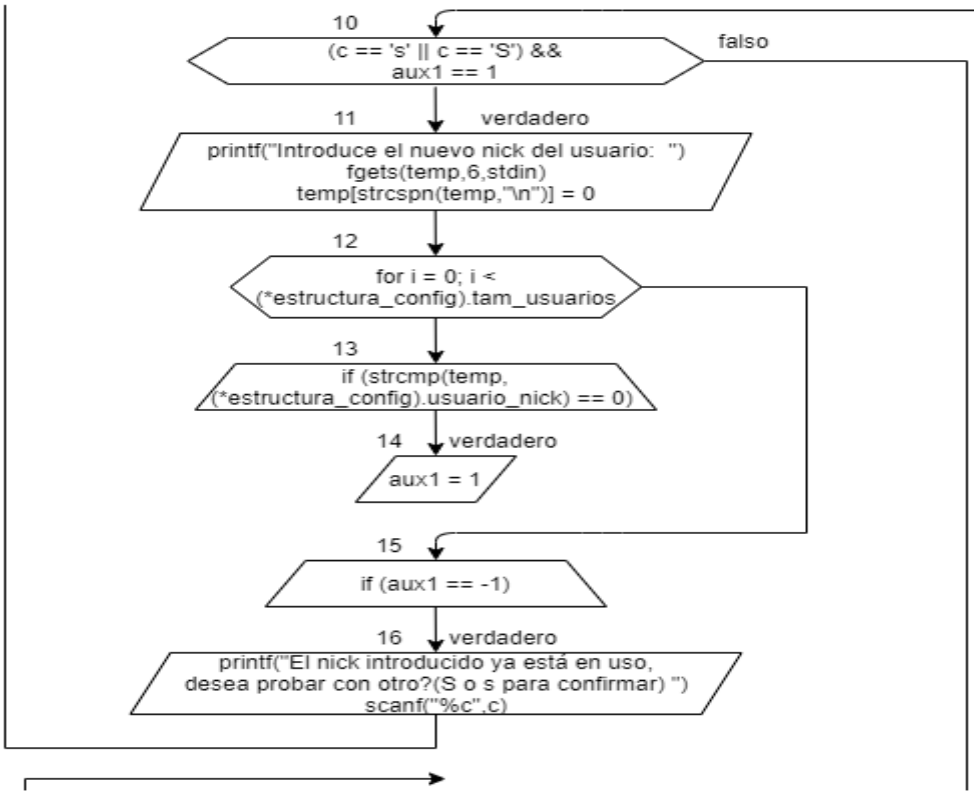
	Devolución		
	Valido	No Valido	Limite
equipo_id	Coincidirá con alguno de la estructura, devuelve aux==1	No coincide con ninguno de la estructura, aux == 0 y entonces salta el error	No coincide con ninguno de la estructura, aux == 0 y entonces alta el error
futbolista_id	Coincide con alguno de los futbolistas del equipo	Informa al usuario de que no existe futbolista con ese ID	
valoracion	La valoración es positiva y menor o igual que 10, se aplica la valoración al futbolista	Es un bucle DO WHILE, si es menor de 0 o mayor que 10, vuelve al bucle hasta que sea correcta	Si es 10'5, como hemos tomado de ejemplo, vuelve al bucle
tam_futbolistas	Si el tamaño es valido, entrara en el bucle for del nodo 5, por lo que se recorrerá el bucle	En el caso de que sea 0 o menor, no entrara en el bucle for y entrara al if con aux== 0 por lo que salta error	

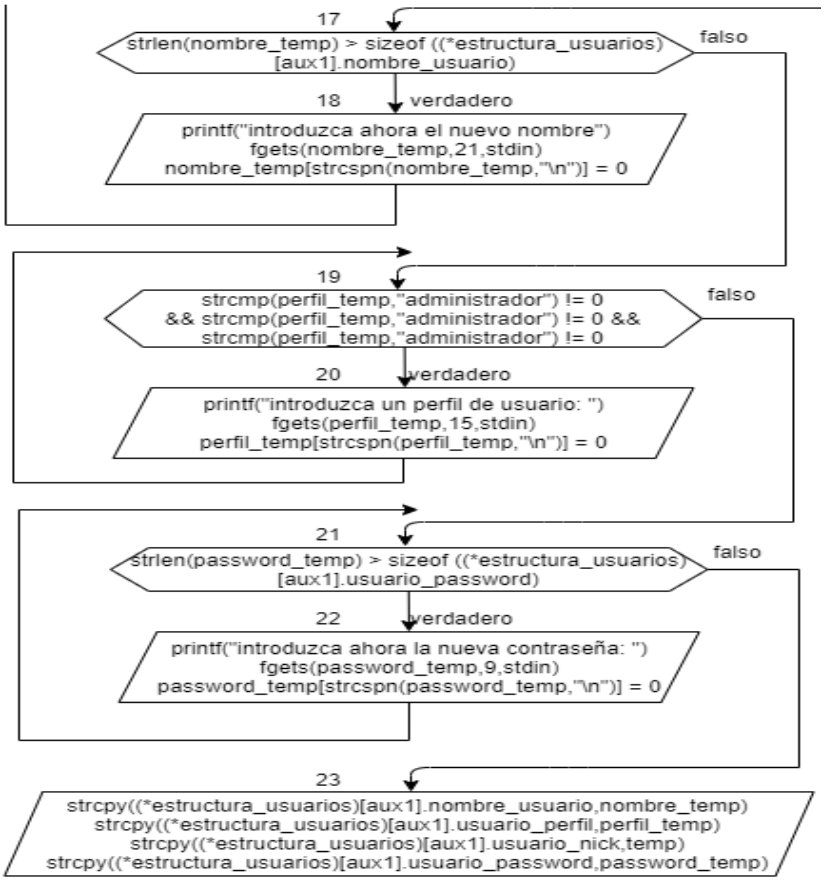
Función de Álvaro Santos Romero: Modificar Usuarios (participante.h)

PRUEBAS DE CAJA BLANCA

1) GRAFO DE FLUJO




10
(c == 's' || c == 'S') &&
aux1 == 1
falso
11
verdadero
printf("Introduce el nuevo nick del usuario: ")
fgets(temp,6,stdin)
temp[strcspn(temp,"\n")] = 0
12
for i = 0; i <
(*estructura_config).tam_usuarios
13
if (strcmp(temp,
(*estructura_config).usuario_nick) == 0)
14
verdadero
aux1 = 1
15
if (aux1 == -1)
16
verdadero
printf("El nick introducido ya está en uso,
desea probar con otro?(S o s para confirmar) ")
scanf("%c",c)


17
strlen(nombre_temp) > sizeof ((*estructura_usuarios)
[aux1].nombre_usuario)
falso
18
verdadero
printf("introduzca ahora el nuevo nombre")
fgets(nombre_temp,21,stdin)
nombre_temp[strcspn(nombre_temp,"\n")] = 0
19
strcmp(perfil_temp,"administrador") != 0
&& strcmp(perfil_temp,"administrador") != 0 &&
strcmp(perfil_temp,"administrador") != 0
falso
20
verdadero
printf("introduzca un perfil de usuario: ")
fgets(perfil_temp,15,stdin)
perfil_temp[strcspn(perfil_temp,"\n")] = 0
21
strlen(password_temp) > sizeof ((*estructura_usuarios)
[aux1].usuario_password)
falso
22
verdadero
printf("introduzca ahora la nueva contraseña: ")
fgets(password_temp,9,stdin)
password_temp[strcspn(password_temp,"\n")] = 0
23
strcpy((*estructura_usuarios)[aux1].nombre_usuario,nombre_temp)
strcpy((*estructura_usuarios)[aux1].usuario_perfil,perfil_temp)
strcpy((*estructura_usuarios)[aux1].usuario_nick,temp)
strcpy((*estructura_usuarios)[aux1].usuario_password,password_temp)

2) CÁLCULO DE LA COMPLEJIDAD CICLOMÁTICA

CC = nodos predicado + 1 = 5 + 1 = 6

Veamos los posibles caminos:

1. 1-2-3-4-5-8-9 :Introducir nick de usuario que no está en la estructura
2. 1-2-3-4-6-7-2-10-11-12-15-16 :Introducir nuevo nick de usuario ya existente
3. 1-2-3-4-6-7-2-10-11-12-13-14-10-17-18 :Nombre inválido
4. 1-2-3-4-6-7-2-10-11-12-13-14-10-17-18-17-19-20-19 :Perfil no coincide
5. 1-2-3-4-6-7-2-10-11-12-13-14-10-17-18-17-19-20-19-21-22-21 : Contraseña inválida
6. 1-2-3-4-6-7-2-10-11-12-13-14-10-17-18-17-19-20-19-21-22-21-23: Ejecución OK

PRUEBAS DE CAJA NEGRA

Análisis de los valores límites Método modificarUsuarios

INTRODUCIR NICK DE USUARIO

Tipo de entrada	Nº de clases válidas	Nº de clases inválidas
Cadena de 6 caracteres	**2**: cadenas de 1,6 caracteres	**2**: cadenas de 0, 7 caracteres

Entrada: Cadena de 1 carácter = ‘h’
Procedimiento: Se comprueba que ‘h’ no es mayor que 6 y que no existe
Salida: Nick en rango

Entrada: Cadena de 1 carácter = ‘h’
Procedimiento: Se comprueba que ‘h’ no es mayor que 6 y que existe
Salida: Nick no disponible

Entrada: Cadena de 1 carácter = ‘Robert’’
Procedimiento: Se comprueba que ‘Robert’’ no es mayor que 6 y que no existe
Salida: Nick en rango

Entrada: Cadena de 1 carácter = ‘Robert’’
Procedimiento: Se comprueba que ‘Robert’’ no es mayor que 6 y que no existe
Salida: Nick no disponible

Entrada: Cadena de 0 carácter = ‘’
Procedimiento: El programa sigue pidiendo al usuario que introduzca una cadena

Salida:

Entrada: Cadena de 7 carácter = ‘Roberto’’
Procedimiento: Se comprueba que ‘Roberto’ no es mayor que 6 y que no existe
Salida: Nick fuera de rango

INTRODUCIR NICK DE USUARIO 2

Tipo de entrada	Nº de clases válidas	Nº de clases inválidas
Cadena de 6 caracteres	**2**: cadenas de 1,6 caracteres	**2**: cadenas de 0, 7 caracteres

(Mismo procedimiento que introducir usuario 1)

Entrada: Cadena de 1 carácter = ‘h’
Procedimiento: Se comprueba que ‘h’ no es mayor que 6 y que no existe
Salida: Nick en rango

Entrada: Cadena de 1 carácter = ‘h’
Procedimiento: Se comprueba que ‘h’ no es mayor que 6 y que existe
Salida: Nick no disponible

Entrada: Cadena de 1 carácter = ‘Robert’’
Procedimiento: Se comprueba que ‘Robert’’ no es mayor que 6 y que no existe
Salida: Nick en rango

Entrada: Cadena de 1 carácter = ‘Robert’’
Procedimiento: Se comprueba que ‘Robert’’ no es mayor que 6 y que no existe
Salida: Nick no disponible

Entrada: Cadena de 0 carácter = ‘’’
Procedimiento: El programa sigue pidiendo al usuario que introduzca una cadena
Salida: Bucle

Entrada: Cadena de 7 carácter = ‘Roberto’’
Procedimiento: Se comprueba que ‘Roberto’ no es mayor que 6 y que no existe
Salida: Nick fuera de rango

INTRODUCIR NUEVO NOMBRE

Tipo de entrada	Nº de clases válidas	Nº de clases inválidas
Cadena de 21 caracteres	2: cadenas de 1,21 caracteres	2: cadenas de 0, 22 caracteres

Entrada: Cadena de 1 carácter = ‘h’
Procedimiento: Se comprueba que ‘h’ no es mayor que 21
Salida: nombre en rango

Entrada: Cadena de 21 carácter = ‘Manuel Butanos’
Procedimiento: Se comprueba que ‘Manuel Butanos’ no es mayor que 21
Salida: Nombre en rango

Entrada: Cadena de 0 carácter = ‘’
Procedimiento: El programa sigue pidiendo al usuario que introduzca una cadena
Salida: Bucle

Entrada: Cadena de 22 carácter = ‘Esternocleidomastoideo’
Procedimiento: Se comprueba que ‘Esternocleidomastoideo’’ no es mayor que 21 y que no existe
Salida: Nombre fuera de rango

INTRODUCIR PERFIL

Tipo de entrada	Nº de clases válidas	Nº de clases inválidas
Cadena de 15 caracteres	3: cadenas equivalentes a: "participante", "administrador" o "cronista"	2: cadenas de 0, 22 caracteres

Entrada: Cadena de 15 carácter = ‘participante’
Procedimiento: Se comprueba que ‘participante’ coincide con las cadenas propuestas
Salida: nombre en rango

Entrada: Cadena de 15 carácter = ‘administrador’
Procedimiento: Se comprueba que ‘administrador’ coincide con las cadenas propuestas
Salida: nombre en rango

Entrada: Cadena de 15 carácter = ‘cronista’
Procedimiento: Se comprueba que ‘cronista’ coincide con las cadenas propuestas
Salida: nombre en rango

Entrada: Cadena de 0 carácter = ‘’
Procedimiento: El programa sigue pidiendo al usuario que introduzca una cadena
Salida: Bucle

Entrada: Cadena 16 carácter = ‘participantes’
Procedimiento: Se comprueba que ‘participantes’ no coincide con las cadenas propuestas
Salida: perfil fuera de rango

INTRODUCIR NUEVA CONTRASEÑA

Tipo de entrada	Nº de clases válidas	Nº de clases inválidas
Cadena de 9 caracteres	**2**: cadenas de 1,9 caracteres	**2**: cadenas de 0, 10 caracteres

Entrada: Cadena de 1 carácter = ‘u’
Procedimiento: Se comprueba que ‘u’ no es mayor que 9
Salida: contraseña en rango

Entrada: Cadena de 9 carácter = ‘ubuntu88’
Procedimiento: Se comprueba que ‘ubuntu88’ no es mayor que 9
Salida: contraseña en rango

Entrada: Cadena de 0 carácter = ‘’
Procedimiento: El programa sigue pidiendo al usuario que introduzca una cadena
Salida: Bucle

Entrada: Cadena de 10 carácter = ‘bababoy33’
Procedimiento: Se comprueba que ‘bababoy33’ no es mayor que 9
Salida: contraseña fuera de rango