Escuela Superior de Ingeniería
Departamento de Ingeniería Informática

Grado en Ingeniería Informática

METODOLOGÍA DE LA PROGRAMACIÓN

Documentación Liga Fantástica.

Escuela Superior de Ingeniería, Av. Universidad de Cádiz nº 10, 11519, Puerto Real, Cádiz.

Antonio Roldán Andrade

Marcos Pérez Fernández

Juan Carlos de León Amores

Índice:

1. Funcionamiento del programa.

La Liga Fantástica es un programa que simula un juego de fútbol online desde el cual, cualquier participante registrado podrá crear sus propias plantillas de jugadores reales en base a un presupuesto que estará establecido por un administrador del juego. Después de cada jornada de fútbol de la vida real, los cronistas del juego asignarán una puntuación a los jugadores de fútbol según el rendimiento que hayan mostrado en la vida real, que irá del 1 al 10. En base a esta puntuación los participantes tendrán en sus plantillas una valoración global de plantilla con la que competirán con otros participantes online.

El programa dispondrá de tres tipos de perfiles de usuarios:

- Un perfil de **participante**: tendrá la posibilidad de crear sus propias plantillas de fútbol en base a un presupuesto previamente establecido y competir en la Liga Fantástica contra los demás participantes.
- Un perfil de **administrador**: realizará las tareas de configuración del juego, como por ejemplo la asignación de presupuestos para fichajes, el límite de plantillas por participantes, etc.
- Un perfil de **cronista**: realizará las valoraciones de jugadores de la vida real según el rendimiento que den estos en cada partido de cada jornada.

Todos los datos de la Liga Fantástica estarán almacenados en ficheros para poder conservar toda la información y volverla a utilizar en posteriores ejecuciones del programa. Al iniciar el programa, dicha información se volcará desde los ficheros a estructuras de datos en memoria y al terminar la ejecución del programa la información se actualizará en los ficheros. De esta manera todo el funcionamiento de la plataforma tendrá lugar en memoria principal.

2. Descomposición de la plataforma en módulos.

Al analizar el funcionamiento del programa, decidimos en primer lugar que habría un módulo *core* en donde estarían todas las funciones necesarias para transferir todos los datos de los ficheros de texto a las estructuras correspondientes en memoria principal. Las funciones de este módulo también escribirían la información resultante desde la memoria a los ficheros de texto.

El siguiente módulo sería *participante.* En él se encuentran todas las funciones que nos permiten navegar por los diferentes menús de participante, registrarse, acceder al sistema, crear una plantilla, configurarla, y competir en la liga.

El tercer módulo es *cronista,* donde incluimos las funciones necesarias para que el cronista pueda acceder al sistema, valorar jugadores y plantillas del juego.

Por último, el módulo *administrador*, en el que se encuentran las funciones que permiten la modificación de los datos del juego, como por ejemplo el presupuesto de los participantes a la hora de realizar fichajes.

3. Módulo *core*.

Al principio de la ejecución del programa, las funciones de lectura a las que hemos llamado “recovery” abren sus respectivos ficheros de texto especificados en el propio código de cada función de lectura y cargan los datos en memoria.

Por otra parte, las funciones de escritura a las que hemos llamado “update” se encargan del proceso inverso. Una vez realizados cambios en los datos almacenados en las estructuras, estas funciones abren el fichero de texto correspondiente y escriben todos los datos sustituyendo a los anteriores. De esta forma hemos asegurado la persistencia de los datos para futuras ejecuciones del programa.

4. Módulo *participante*.

Contiene una función para mostrar un menú de *participante*, en el que se anidarán otras funciones que permitirán hacer lo siguiente:

- Crear plantillas: *void part_crear_plantilla(int);*
- Configurar plantillas: *void part_config_plantilla(int);*
- Listar plantillas: *void part_list_plantilla(int);*
- Eliminar plantillas: *void part_eliminar_plantilla(int);*
- Ver ranking de puntuaciones de la Liga Fantástica: *void part_ranking(int);*
- Salir de la sesión: *void part_exit();*

5. Módulo *cronista*.

Contiene una función para mostrar un menú de *cronista*, en el que se anidarán otras funciones que permitirán hacer lo siguiente:

- Listar todos los equipos del juego: *void cro_listar_equipos();*
- Valorar jugadores que estén en un equipo: *void cro_valorar_equipos();*

6. Módulo *admin.*

Contiene una función para mostrar un menú de *administrador*, en el que se anidarán otras funciones que permitirán hacer lo siguiente:

- Crear nuevo equipo en el juego: *void adm_new_team();*
- Listar todos los equipos del juego: *void adm_list_teams();*

7. Ejecución del programa.

Al ejecutar *Liga Fantástica*, el programa nos pedirá que iniciemos sesión con un usuario y una contraseña, a través de un pequeño menú diseñado en la función principal *main,* en el cual se llamarán a los distintos menús de los demás módulos, dependiendo de si accedemos al programa con uno de los siguientes tres roles: administrador, participante o cronista. En función de esto, visualizaremos un menú u otro.

8. Pruebas.

- **Módulo *admin.***

➢ Prueba de caja negra:

Probaremos la función *void adm_new_team(); .*

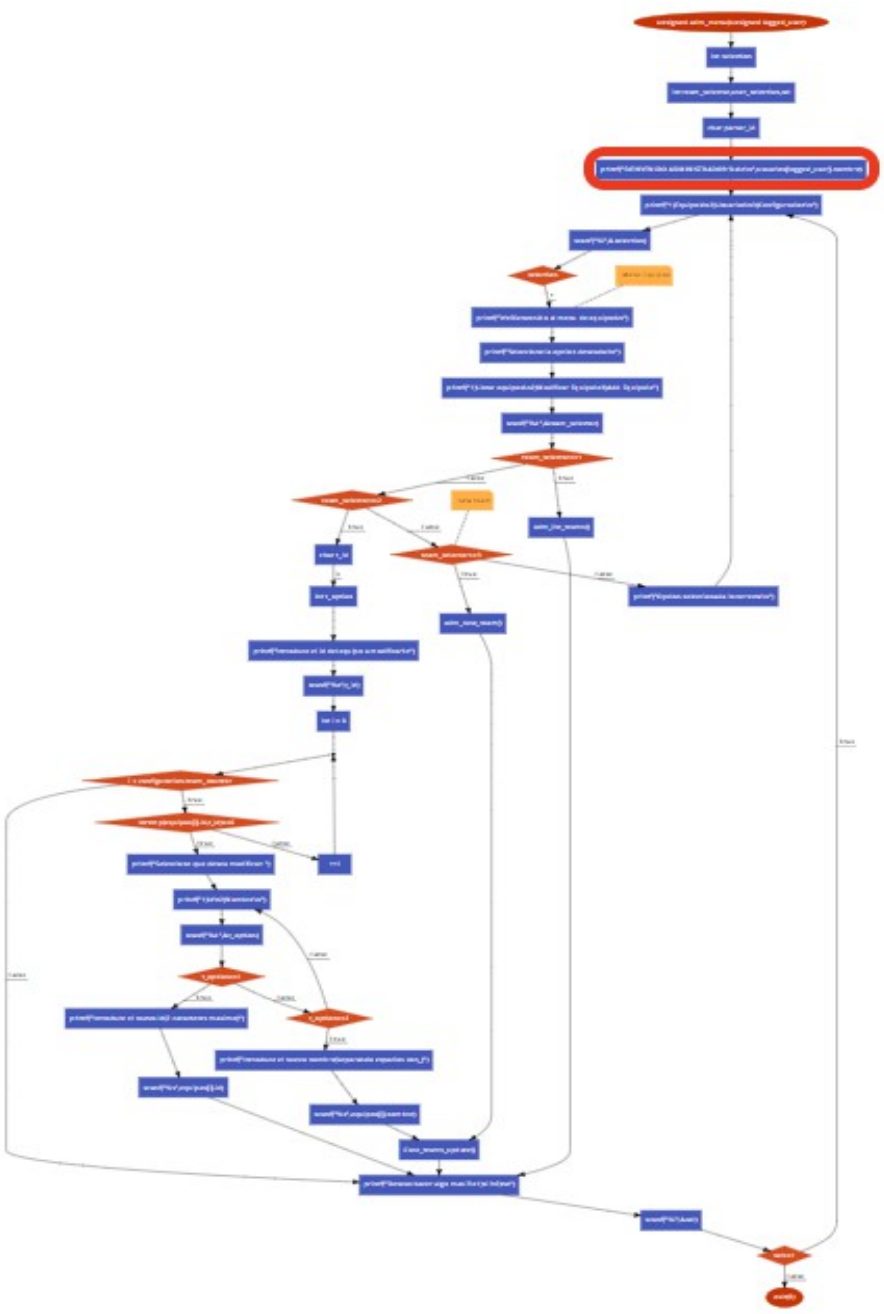
```
void adm_new_team(){
    TEAMFILE = fopen("a","data/Equipos.txt");
    assert(TEAMFILE!=NULL);

    team temp_team;
    printf("Introduce el id\n");
    scanf("%s",temp_team.id);
    printf("Introduce el nombre\n");
    scanf("%s",temp_team.nombre);
    fprintf(TEAMFILE,"%s",temp_team.id);
    fprintf(TEAMFILE,"%c",'\n');
    fprintf(TEAMFILE,"%s",temp_team.nombre);
    fprintf(TEAMFILE,"%c",'\n');
    fclose(TEAMFILE);
}
```

● **ENTRADA**	● **SALIDA**
● El fichero tiene un problema a la hora de acceder a él o simplemente no existe.	● El programa no continúa con la operación gracias a assert(TEAMFILE!=NULL).
● Introduzco los datos del nuevo equipo.	● Se escriben dichos parámetros en el fichero TEAMFILE.

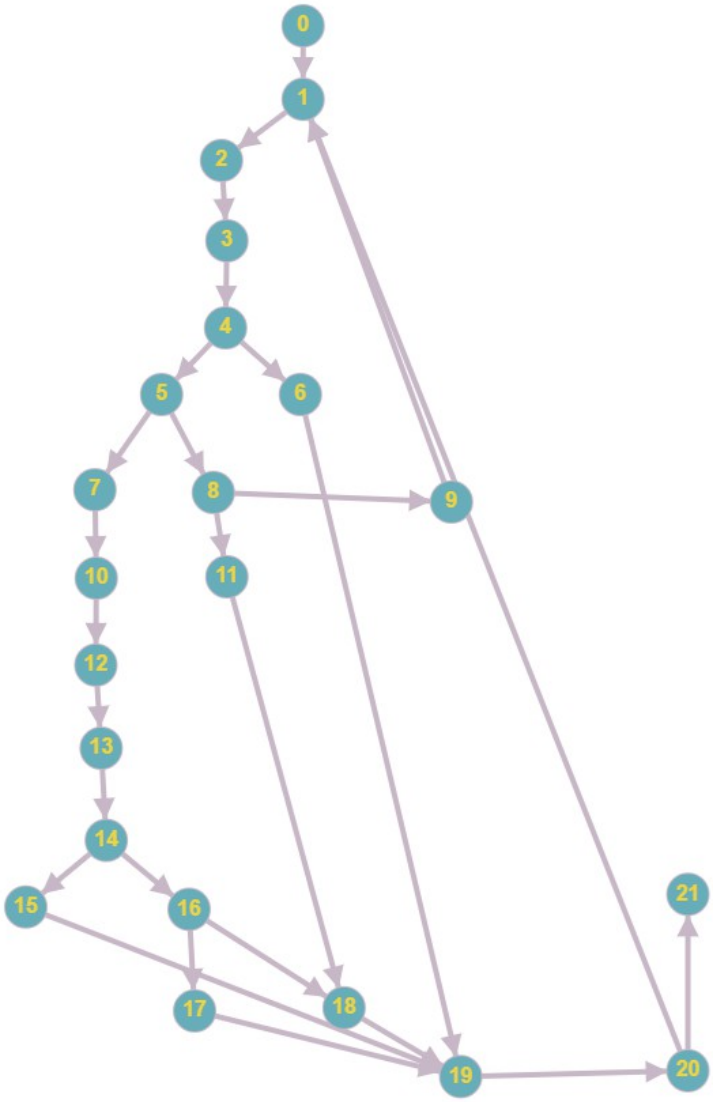
Se comprueba dicho cometido de la función al introducir un nuevo equipo en su fichero correspondiente, teniendo en cuenta previamente el usuario, la característica de ¨administrador¨.

- Prueba de caja blanca:
  - <u>Prueba de ruta básica.</u>

    Probaremos la función de *unsigned adm_menu(unsigned logged_user)*. Su diagrama de flujo sería de la siguiente forma:



Su grafo simplificado correspondiente sería:



Al calcular su complejidad ciclomática da los siguientes resultados:

V(G) = NNP + 1 = 9

V(G) = NA - NN + 2 = 9

Al conocerse el número de rutas independientes, se mostrarán en la siguiente tabla:

RUTA BÁSICA	ORDEN
1	0, 1, 2, 3, 4, 5, 7, 10, 12, 13, 14, 15, 19, 20, 21
2	0, 1, 2, 3, 4, 5, 7, 10, 12, 13, 14, 16, 17, 19, 20, 21
3	0, 1, 2, 3, 4, 5, 7, 10, 12, 13, 14, 16, 18, 19, 20, 21
4	0, 1, 2, 3, 4, 5, 8, 11, 18, 19, 20, 21
5	0, 1, 2, 3, 4, 5, 8, 9, 1… (Continua como la ruta 1)
6	0, 1, 2, 3, 4, 6, 18, 19, 20, 1… (Continua como en la ruta 1)
7	0, 1, 2, 3, 4, 5, 8, 9, 1… (Continua como la ruta 2)
8	0, 1, 2, 3, 4, 6, 18, 19, 20, 1… (Continua como en la ruta 2)
9	0, 1, 2, 3, 4, 6, 19, 20, 21

- Prueba de bucle.

Se realizará sobre el siguiente fragmento de código:

```
for (int i = 0; i <configuration.user_counter ; ++i) {
    printf("ID=>%s\tNombre=>%s\n",usuarios[i].id,usuarios[i].nombre);
    printf("NameTag=>%s\tRole=>%s\n",usuarios[i].name_tag,usuarios[i].role);
}
```

Precondiciones	Acciones	Después de iterar	Deseado
i = configuartion.user_counter	……..	i es igual a la variable sobre la que se itera	No entra en el bucle. Ocurre el resultado esperado
i = 0	i++ (Primera iteración)	i = 1	Imprime por pantalla los datos del usuario 1
i = 1	i++ (Segunda iteración)	i = 2	Imprime por pantalla los datos del usuario 2
i = configuartion.user_counter - 1	i++ (Última iteración)	i = configuartion.user_counter	Imprime los datos del último usuario
i = configuartion.user_counter + 1	……..	i = configuartion.user_counter + 1	No entra en el bucle.

- **Módulo *participante.***

  - Prueba de caja negra:

    Probaremos la función *void part_list_plantilla(int);* .

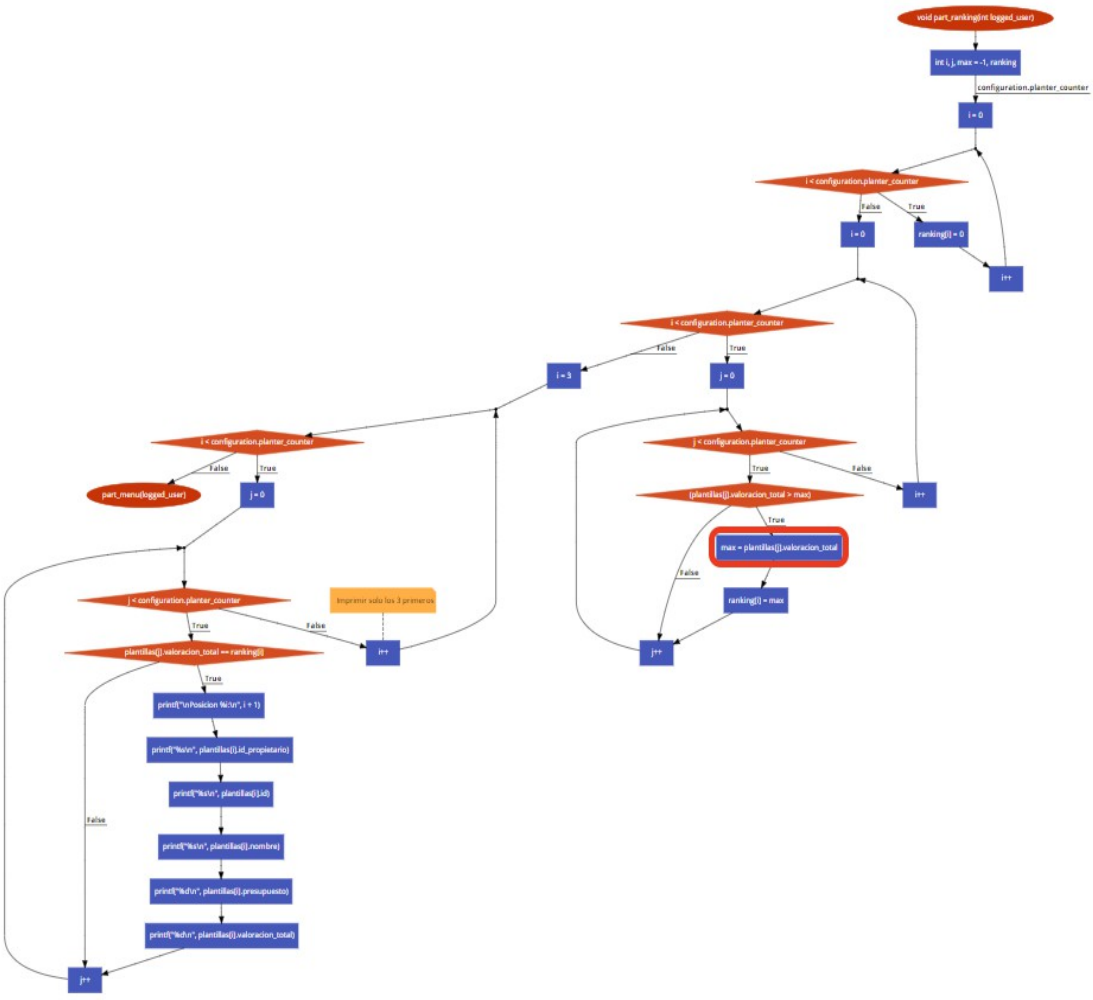
```
void part_list_plantilla(int logged_user){
    int i, counter = 0;
    printf("\tListado de todas sus plantillas:\n");

    for (i=0; i<configuration.planter_counter ; ++i) {
        if(strcmp(usuarios[logged_user].id, plantillas[i].id_propietario)==0) {
            counter++;
            printf("%s\n", plantillas[i].id);
            printf("%s\n", plantillas[i].nombre);
            printf("%d\n", plantillas[i].valoracion_total);
        }
    }
    if(counter==0){
        printf("Usted no posee ninguna plantilla");
        part_menu(logged_user);
    }
    part_menu(logged_user);
}
```

ENTRADA	SALIDA
Recibe la id del usuario.	Imprime por pantalla los datos de sus plantillas.
El usuario posee al menos una plantilla.	Imprime por pantalla dicha plantilla y vuelve al menú.
El usuario no posee ninguna plantilla.	Se le informa de que no posee ninguna y es mandado al menú.

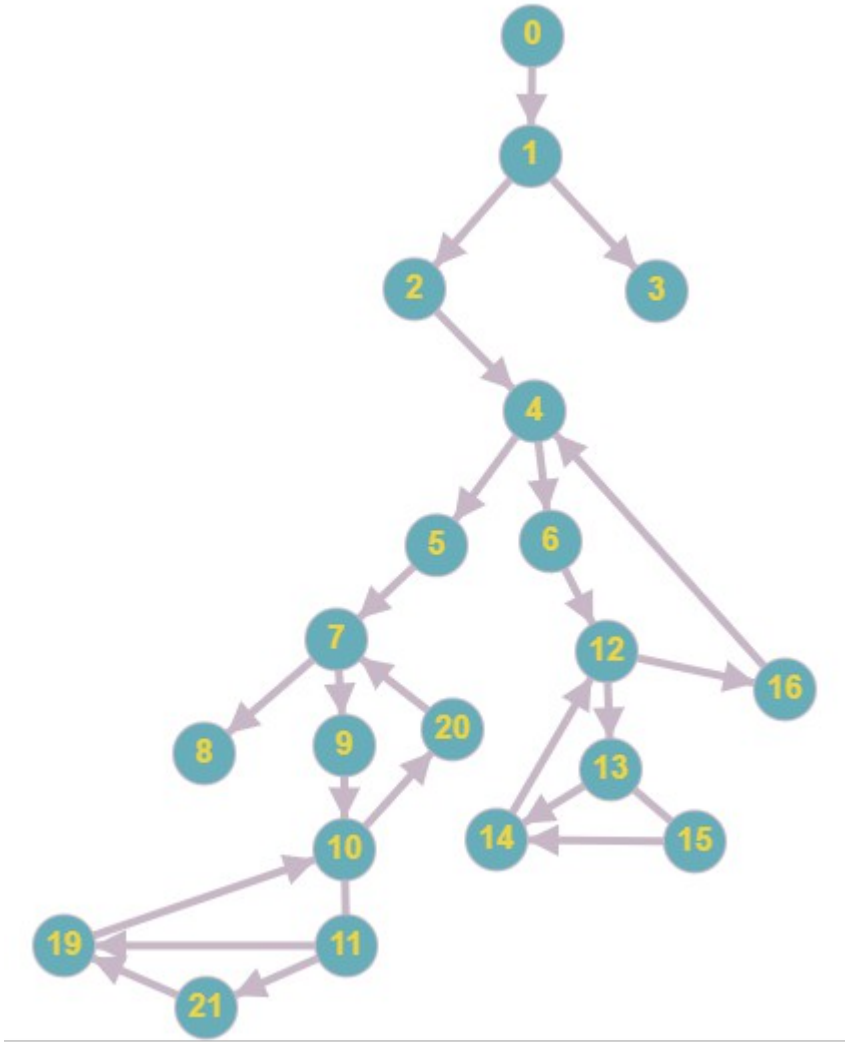
Como se puede observar, la función cumple con su cometido, siempre y cuando el usuaria posea la característica de ¨participante¨.

- Prueba de caja blanca:
  - Prueba de ruta básica.

    Usaremos la función void par_ranking(int); cuyo flujo es:



Su grafo simplificado sería:



Al calcular su complejidad ciclomática da los siguientes resultados:

V(G) = NNP + 1 = 6

V(G) = NA - NN + 2 = 6

Al conocerse el número de rutas independientes, se mostrarán en la siguiente tabla:

RUTA BÁSICA	ORDEN
1	0, 1, 2, 4, 5, 7, 8
2	0, 1, 2, 4, 5, 7, 9, 10, 11, 21, 19, 19, 29, 7, 8
3	0, 1, 2, 4, 5, 6, 9, 10, 11, 19, 10, 20, 7, 8
4	0, 1, 3
5	0, 1, 2, 4, 6, 12, 13, 15, 14, 12, 16, 4… (Continúa como ruta 1)
6	0, 1, 2, 4, 6, 12, 13, 15, 14, 12, 16, 4… (Continúa como ruta 2)

- Prueba de bucle.

Se realizará sobre el siguiente fragmento de código:

```
for (j = 0; j < configuration.planter_counter; j++) {
    if (plantillas[j].valoracion_total == ranking[i]) {
        printf("\nPosicion %i:\n", i + 1);
        printf("%s\n", plantillas[i].id_propietario);
        printf("%s\n", plantillas[i].id);
        printf("%s\n", plantillas[i].nombre);
        printf("%d\n", plantillas[i].presupuesto);
        printf("%d\n", plantillas[i].valoracion_total);
    }
}
```

Precondiciones	Acciones	Después de iterar	Deseado
j = configuartion.planter_counter	……..	j es igual a la variable sobre la que se itera	No entra en el bucle. Ocurre el resultado esperado
j = 0	j++ (Primera iteración) Comprueba que la valoración está en el ranking	j = 1	Imprime por pantalla los datos de la plantilla 1
j = 1	j++ (Segunda iteración) Comprueba que la valoración está en el ranking	j = 2	Imprime por pantalla los datos de la plantilla 2
j = configuartion.planter_counter- 1	j++ (Última iteración) Comprueba que la valoración está en el ranking	j = configuartion.planter_counter	Imprime los datos de la última plantilla
j = configuartion.planter_counter+ 1	……..	j = configuartion.planter_counter + 1	No entra en el bucle.

- **Módulo *core.***

  - Prueba de caja negra:

    Para esta prueba se utilizará la función *unsigned core_login();*

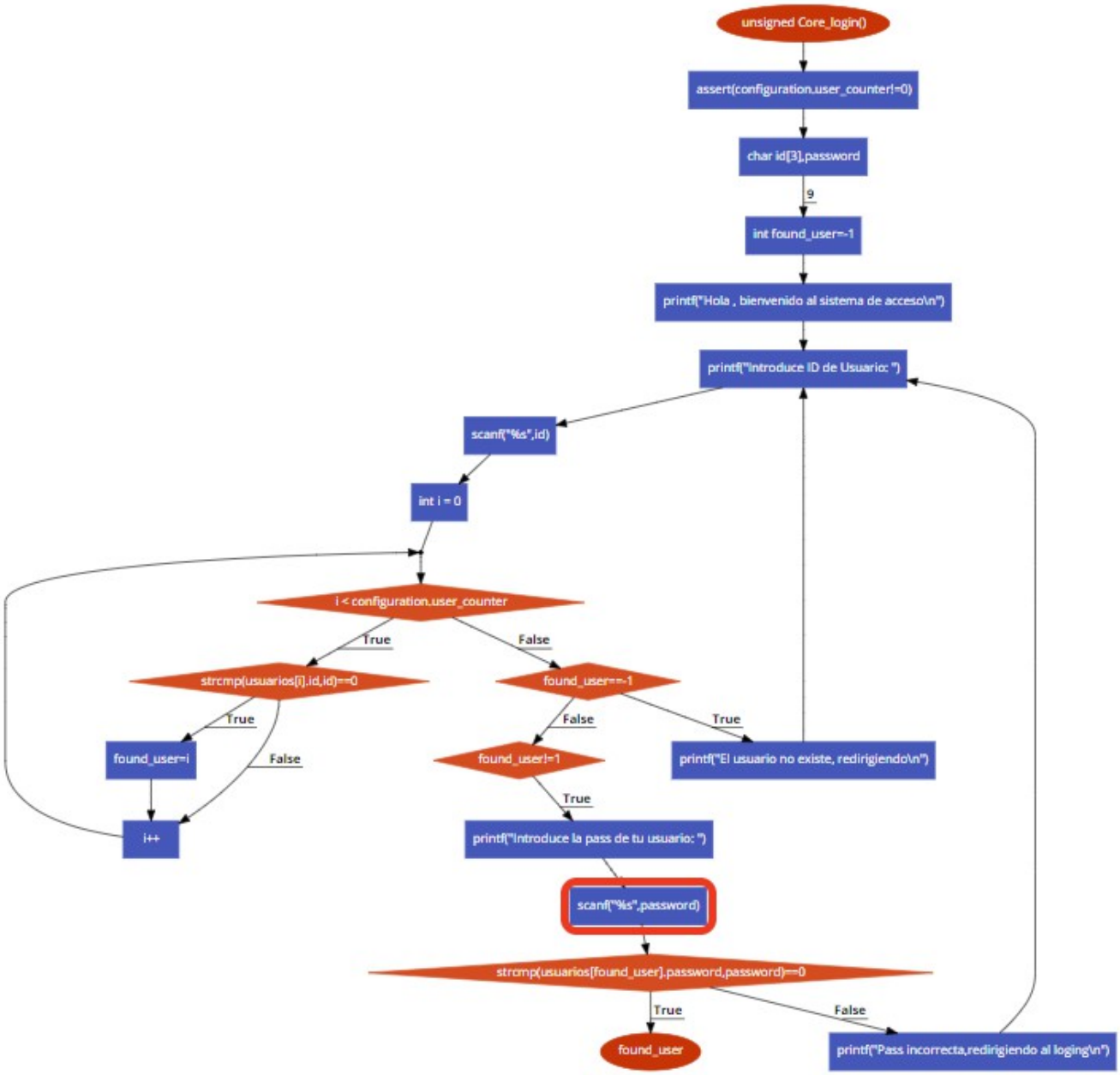
```c
unsigned Core_login() {
    assert(configuration.user_counter!=0);
    char id[3],password[9];
    int found_user=-1;
    printf("Hola , bienvenido al sistema de acceso\n");
    ppio:
    printf("Introduce ID de Usuario: ");
    scanf("%s",id);
    for (int i = 0; i < configuration.user_counter; i++){
        if(strcmp(usuarios[i].id,id)==0)
            found_user=i;
    }
    if(found_user==-1){
        printf("El usuario no existe, redirigiendo\n");
        goto ppio;
    }
    if(found_user!=1){
        printf("Introduce la pass de tu usuario: ");
        scanf("%s",password);
        if(strcmp(usuarios[found_user].password,password)==0)
            return found_user;
        else{
            printf("Pass incorrecta,redirigiendo al loging\n");
            goto ppio;
        }
    }
```

ENTRADA	SALIDA
Se introduce un id que existe y su respectiva contraseña	La función devuelve el usuario encontrado
Se introduce un id que no existe	Imprime por pantalla que dicho usuario no existe y reinicia la consulta
Se introduce id que existe pero la contraseña es incorrecta	Imprime por pantalla la restricción de acceso a esa cuenta y reinicia la consulta

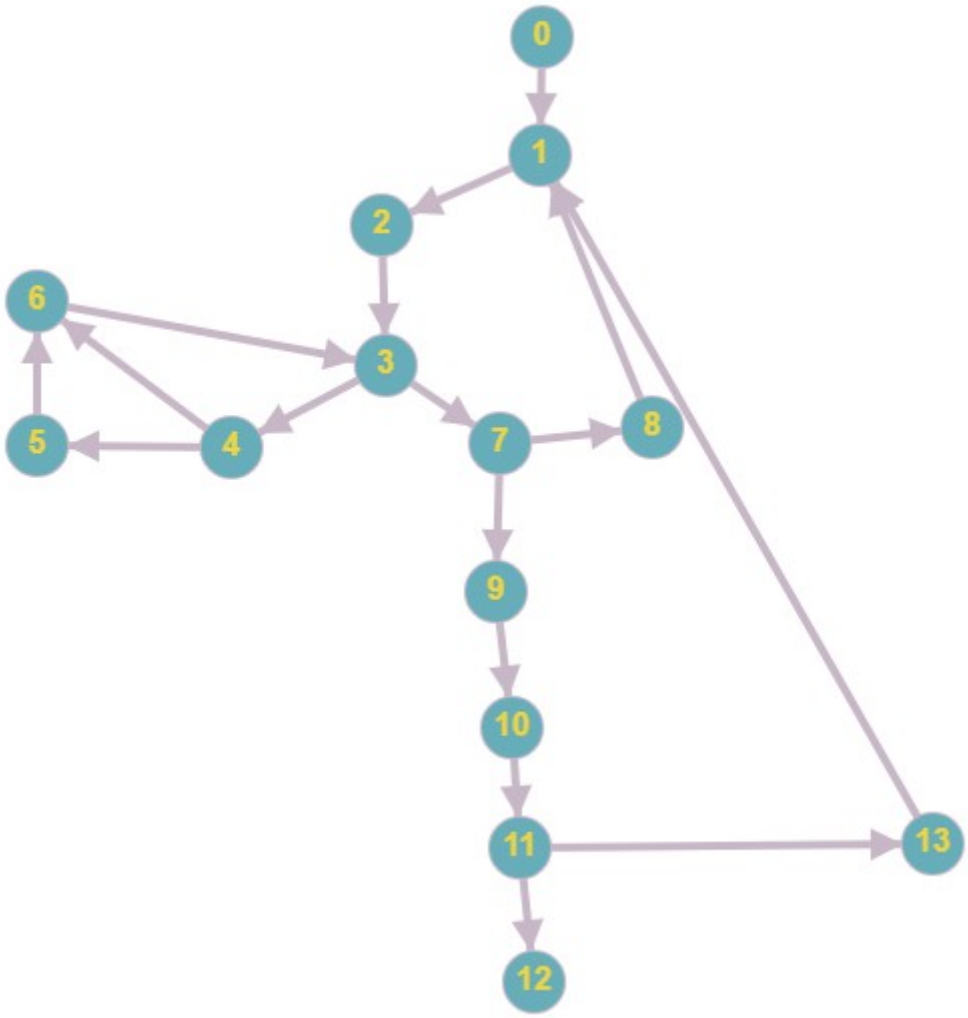
La función realiza sus operaciones como es debido, sin dejar ningún tipo de vulnerabilidad al acceso de cualquier usuario sin el conocimiento de su código identificador y su contraseña.

- Prueba de caja blanca:
  - Prueba de ruta básica.

    Para esta prueba se utilizará la función *unsigned core_login();* cuyo flujo es el siguiente:


unsigned Core_login()
assert(configuration.user_counter!=0)
char id[3],password
9
int found_user=-1
printf("Hola , bienvenido al sistema de acceso\n")
printf("Introduce ID de Usuario: ")
scanf("%s",id)
int i = 0
i < configuration.user_counter
True
False
strcmp(usuarios[i].id,id)==0
found_user==-1
True
False
True
found_user=i
False
found_user!=1
printf("El usuario no existe, redirigiendo\n")
True
i++
printf("Introduce la pass de tu usuario: ")
scanf("%s",password)
strcmp(usuarios[found_user].password,password)==0
True
False
found_user
printf("Pass incorrecta,redirigiendo al loging\n")

Su grafo sería el siguiente:



Al calcular su complejidad ciclomática da los siguientes resultados:

V(G) = NNP + 1 = 5

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

Al conocerse el número de rutas independientes, se mostrarán en la siguiente tabla:

RUTA BÁSICA	ORDEN
1	0, 1, 2, 3, 4, 5, 6, 3, 7, 9, 10, 11, 12
2	0, 1, 2, 3, 7, 9, 10, 11, 13, 1… (Continúa como la ruta 1)
3	0, 1, 2, 3, 7, 8, 1... (Continúa como la ruta 1)
4	0, 1, 2, 3, 4, 6, 3, 7, 9, 10, 11, 12
5	0, 1, 2, 3, 7, 9, 10, 11, 12

- Prueba de bucle.

  Se realiza sobre este fragmento de código:

```
for (int i = 0; i < configuration.user_counter; i++){
    if(strcmp(usuarios[i].id,id)==0)
        found_user=i;
}
```

Precondiciones	Acciones	Después de iterar	Deseado
i = configuartion.user_counter	……..	i es igual a la variable sobre la que se itera	No entra en el bucle. Ocurre el resultado esperado
i = 0	i++ (Primera iteración) Comprueba si el id escrito existe	i = 1	Si son iguales, recoge dicho valor en ¨found_user¨
i = 1	i++ (Segunda iteración) Comprueba si el id escrito existe	i = 2	Si son iguales, recoge dicho valor en ¨found_user¨
i = configuartion.user_counter - 1	i++ (Última iteración) Comprueba si el id escrito existe	i = configuartion.user_counter	Si son iguales, recoge dicho valor en ¨found_user¨
i = configuartion.user_counter + 1	……..	i = configuartion.user_counter + 1	No entra en el bucle.

Como se puede observar, en cada iteración se realiza la misma operación, ya que al recorrer todos los id's existentes, tan solo podrá ser uno el que coincida (o no) con el id introducido.

- **Módulo *cronista.***

  - Prueba de caja negra:

    Para esta prueba se utilizará la función *void cro_valorar_equipos(); .*

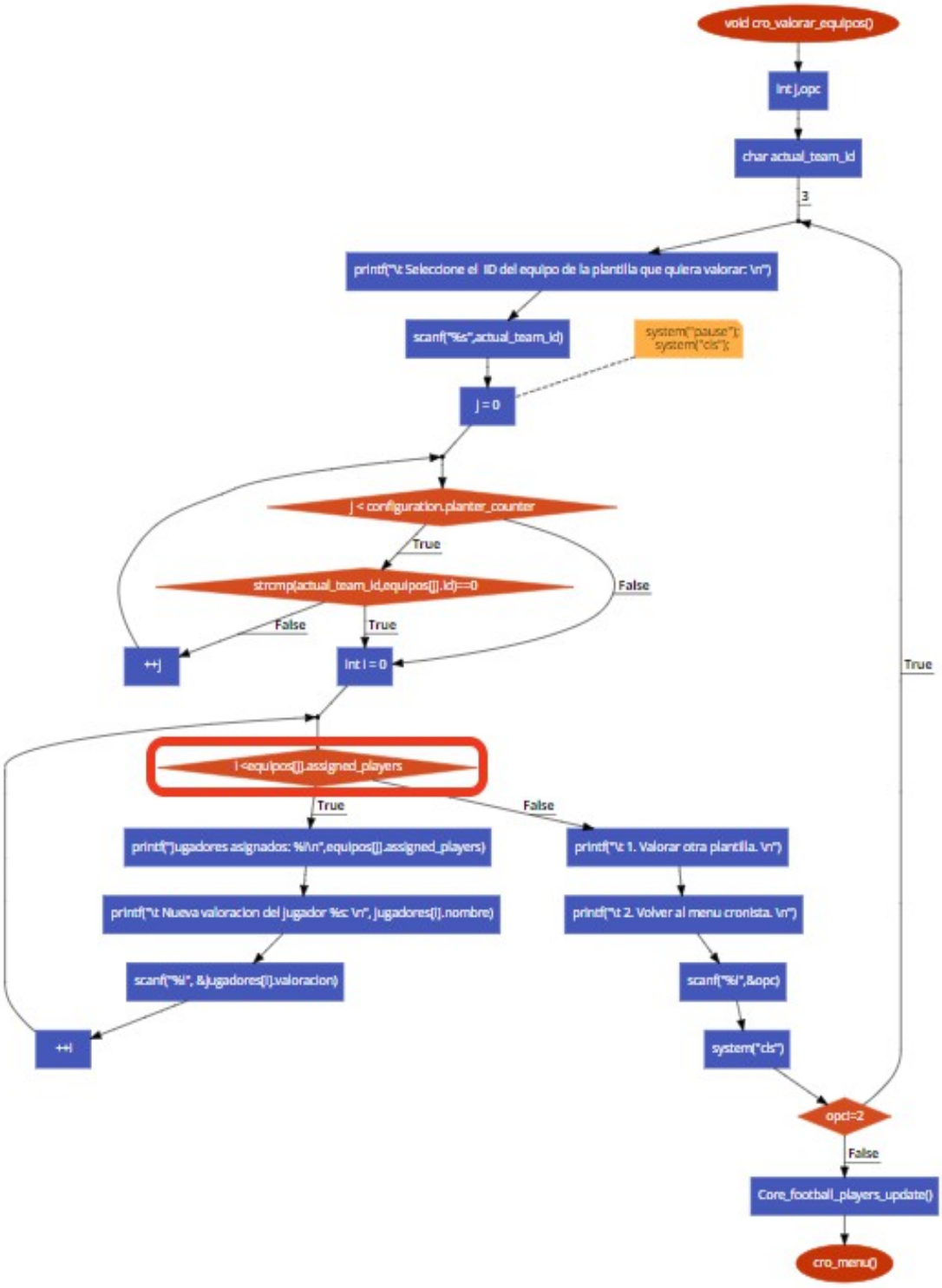
```c
void cro_valorar_equipos()
{
    int j,opc;
    char actual_team_id[3];
    do{
        printf("\t Seleccione el  ID del equipo de la plantilla que quiera valorar: \n");
        scanf("%s",actual_team_id);
        //system("pause");
        //system("cls");
        for (j = 0; j < configuration.planter_counter; ++j) {
            if(strcmp(actual_team_id,equipos[j].id)==0)
            {
                break;
            }
        }
        for (int i = 0; i <equipos[j].assigned_players ; ++i) {
            printf("Jugadores asignados: %i\n",equipos[j].assigned_players);
                printf("\t Nueva valoracion del jugador %s: \n", jugadores[i].nombre);
                scanf("%i", &jugadores[i].valoracion);
        }
        printf("\t 1. Valorar otra plantilla. \n");
        printf("\t 2. Volver al menu cronista. \n");
        scanf("%i",&opc);
        system("cls");
    }while(opc!=2);
    Core_football_players_update();
    cro_menu();
}
```

ENTRADA	SALIDA
Se introduce un id de plantilla que existe	Te permite asignar valoraciones nuevas a jugadores dentro de dicha plantilla
Se introduce un id que no existe	Imprime por pantalla si desea valorar otra plantilla o volver al menú de cronista
Se introduce que se vuelva a valorar plantilla	La función vuelve a ejecutarse a través de un bucle do_while
Se introduce que se quiere volver al menú	La función retorna al menú de cronista saliendo del bucle do_while

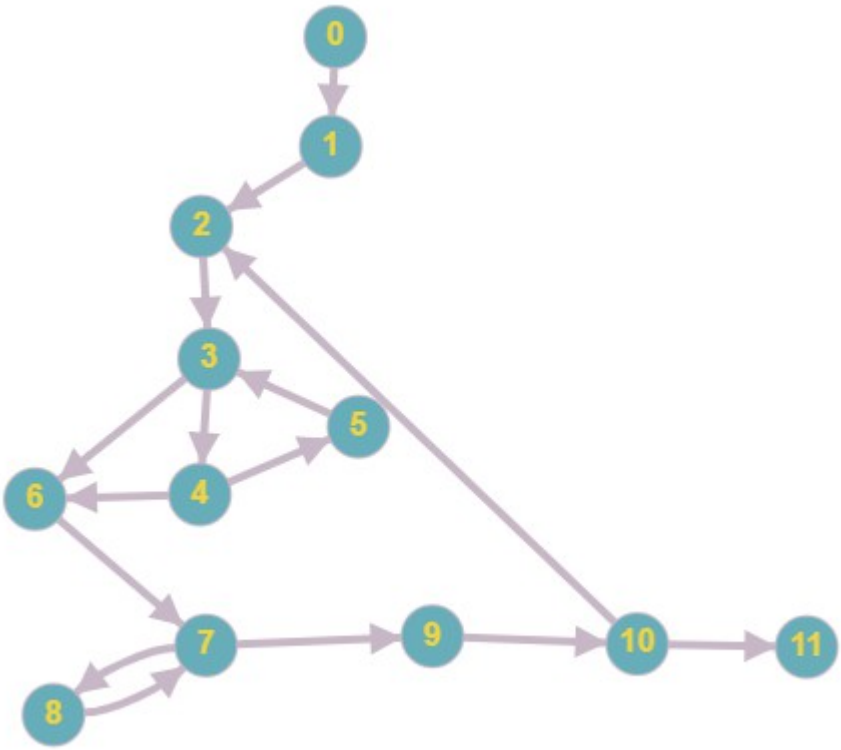
Como se puede observar, la función cumple con las expectativas de su funcionamiento esperado, siempre y cuando se ejecute desde el menú de cronista, siendo el usuario que lo ejecute uno de ellos.

- Prueba de caja blanca.
  - Prueba de ruta básica:

    Utilizaremos la función *void cro_valorar_equipos();* cuyo flujo sería:


void cro_valorar_equipos()
int j,opc
char actual_team_id
3
printf("\t Seleccione el ID del equipo de la plantilla que quiera valorar: \n")
scanf("%s",actual_team_id)
system("pause");
system("cls");
j = 0
j < configuration.planter_counter
True
strcmp(actual_team_id,equipos[j].id)==0
False
False
True
++j
int i = 0
True
i <equipos[j].assigned_players
True
False
printf("Jugadores asignados: %i\n",equipos[j].assigned_players)
printf("\t Nueva valoracion del jugador %s: \n", jugadores[i].nombre)
scanf("%i", &jugadores[i].valoracion)
++i
printf("\t 1. Valorar otra plantilla. \n")
printf("\t 2. Volver al menu cronista. \n")
scanf("%i",&opc)
system("cls")
True
opc!=2
False
Core_football_players_update()
cro_menu()

Su grafo simplificado sería el siguiente:



Al calcular su complejidad ciclomática da los siguientes resultados:

V(G) = NNP + 1 = 5

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

Al conocerse el número de rutas independientes, se mostrarán en la siguiente tabla:

RUTA BÁSICA	ORDEN
1	0, 1, 2, 3, 6, 7, 9, 10, 11
2	0, 1, 2, 3, 4, 5, 3, 6, 7, 8, 7, 9, 10, 11
3	0, 1, 2, 3, 6, 7, 9, 10, 2… (Sigue como la ruta 1)
4	0, 1, 2, 3, 6, 7, 8, 7, 9, 10, 11
5	0, 1, 2, 3, 4, 6, 7, 9, 10, 11

- Prueba de bucle:

  Se realizará sobre el siguiente fragmento de código:

```
for (int i = 0; i <equipos[j].assigned_players ; ++i) {
    printf("Jugadores asignados: %i\n",equipos[j].assigned_players);
        printf("\t Nueva valoracion del jugador %s: \n", jugadores[i].nombre);
        scanf("%i", &jugadores[i].valoracion);
}
```

Precondiciones	Acciones	Después de iterar	Deseado
i = equipos[j].assigned_players	……..	i es igual a la variable sobre la que se itera	No entra en el bucle. Ocurre el resultado esperado
i = 0	i++ (Primera iteración)	i = 1	Asigna valoración nueva al jugador 1
i = 1	i++ (Segunda iteración)	i = 2	Asigna valoración nueva al jugador 2
i = equipos[j].assigned_players - 1	i++ (Última iteración)	i = equipos[j].assigned_players	Asigna valoración nueva al último jugador de la plantilla
i = equipos[j].assigned_players + 1	……..	i = equipos[j].assigned_players + 1	No entra en el bucle.