Metodologia i tecnologia de la Programació
Curs 2014/15

Pràctica 3
Rally LSMaker

	Login	Nom	Grup
Alumnes	LS31343	Manel Manchón Gascó	A
	LS31293	Noa Durán Plass	A

	Segell
Entrega	

Data	31.07.2015

Metodologia i tecnologia de la Programació
Curs 2014/15

Pràctica 3
Rally LSMaker

	Login	Nom	Grup
Alumnes	LS31343	Manel Manchón Gascó	A
	LS31293	Noa Durán Plass	A

	Segell
Entrega	

Data	31.07.2015

Còpia pels alumne

Índex

1. Resum dels enunciats

Fase 1

Degut a l'augment de competicions realitzades amb el robot *Ls Maker*, s'ha decidit incloure una nova modalitat perquè tots aquells que disposin d'un puguin comparar les capacitats dels seus robots: el *Rally JH-MF*. Aquesta nova modalitat representarà una cursa de Rally, en la qual, els LS Maker hauran de mostrar per pantalla els comentaris que els "guiaran" quan es moguin, i a més a més hauran d'executar les comandes incloses en el fitxer ".txt" que se li carregarà al robot abans de començar.

Fase 2

La popularitat de les competicions de Rally han arribat a tal punt, que l'empresa LSDiversiones ha demanat al departament d'informàtica de la universitat que desenvolupi un joc d'estil *retro,* fet a partir de línia de comandes. El joc tractarà essencialment la temàtica de cursa de Rally, en el qual els pilots podran participar en diferents curses i finalment en el campionat.

Al executar el joc es carregarà un fitxer que contindrà una sèrie de pilots i les seves estadístiques, i un fitxer que contindrà la informació referent als circuits disponibles. Aquests fitxers però, poden estar buits, per el quals el joc dispondrà de les opcions d'afegir tant circuits com pilots. Per altra banda, l'usuari contarà amb la possibilitat d'eliminar circuits i pilots al menú principal.

Un cop es disposi de circuits i pilots, hi haurà dos modalitats de joc: la cursa o el campionat. En cas de la cursa, tots els pilots participaran en una única cursa en un circuit escollit per l'usuari.

Per altra banda al campionat es donaran una sèrie de curses en diferents circuits en les quals hi participaran com a mínim un pilot.

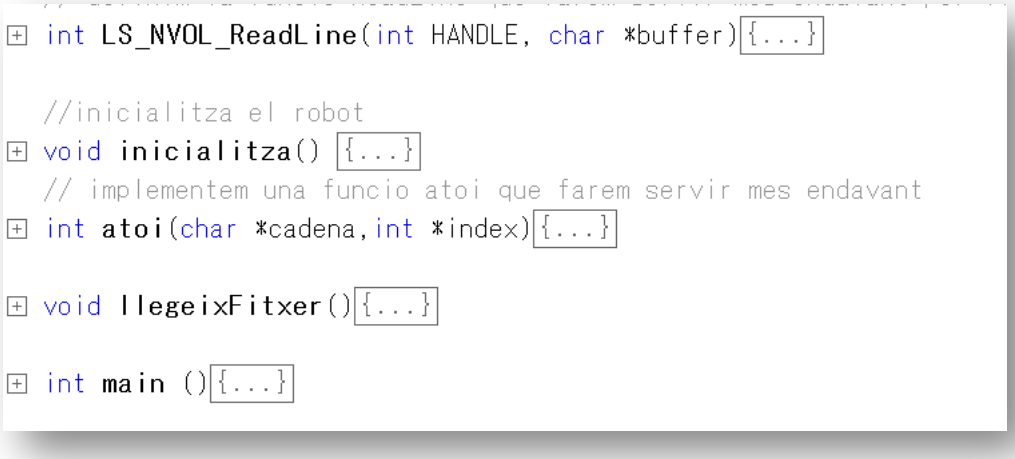
Per acabar el programa contarà amb les opcions de mostrar la informació de tots els pilots disponibles i la opció de realitzar el càlcul d'unes estadístiques que mostraran les millors puntuacions de cada pilot a cada circuit.

2. Disseny

Fase 1

Explicació de les funcions principals:

A l'hora de implementar el codi per el funcionament de l'LS Maker d'acord a les especificacions del circuit, hem fet servir un total de cinc funcions o procediments incloent- hi el *main.*



Funció "LS_NVOL_ReadLine":

L'objectiu principal d'aquesta funció és la de facilitar la lectura de l'arxiu ".txt" que s'ha obert en mode de lectura. Aquesta funció anirà llegint el fitxer lletra per lletra i guardant la informació en una cadena de caràcters que després es passarà per referència. Un cop arriba a un '\n' retorna la cadena omplerta amb la informació llegida. I es col·loca en la següent posició per així poder llegir la següent línia de comandes la següent vegada que es truqui a la funció.

```
int LS_NVOL_ReadLine (int HANDLE, char *buffer){
```

Procediment "inicialitza:"

Aquest procediment és l'encarregat de posar en marxa el funcionament del robot per a que executi les comandes. Aqui es truca al procediment "*LS_Init*" , inclòs en el codi proporcionat per els becaris de la assignatura, el qual s'encarrega d'engegar el robot.

```
void inicialitza() {
```

Funció "atoi":

Aquesta funció, com el seu nom indica, és la funció encarregada d' implementar *atoi*'s, és a dir, transformar caràcters en enters. Mitjançant la implementació d'aquesta funció, s'ha aprofitat per, a l'hora de llegir les variables de la cadena obtinguda arrel del "LS_NVOL_ReadLine", anar transformant els valors directament en enters.

```
int atoi (char *cadena, int *index){
```

Procediment "LlegeixFitxer":

En aquest procediment és on es centren les accions que s'han d'efectuar en aquesta pràctica. Quan el sistema accedeix al procediment, s'obre el fitxer de text en mode de lectura, i entra en un bucle fins que al llegir el fitxer aquest arribi al final. En la primera part d'aquest bucle, el programa es dedica a emmagatzemar els elements corresponents al missatge que s'haurà de pintar per la pantalla del LS Maker. Primerament es transformen les coordenades on s'haurà de pintar el missatge a enters per així després, com s'ha mencionat anteriorment, escriure el missatge a la pantalla.

Seguidament es llegeix la següent línia del fitxer, la qual conté els elements que determinaran la comanda que executarà el LS Maker. Segons la primera lletra de la comanda, el LS Maker es mourà cap a endavant, cap a l'esquerra o la dreta. Per poder realitzar aquesta tasca s'hauran de transformar els elements que representen la velocitat, i, segons la lletra, el temps o els grau en enters. Un cop obtinguts aquests valors, es truquen a les funcions implícites al codi, les quals s'encarregaran d'executar els moviments del robot.

```
void LlegeixFitxer (){
```

Procediment "main":

El procediment "main" és la base d'aquesta fase. Un cop el sistema encén el robot, es generà un bucle infinit que no avançarà fins que l'usuari no premi el "*GpButton"* que es troba al LS_Maker. Un cop l'usuari l'hagui premut, s'executarà el procediment "LlegeixFitxer" i, un cop surti, el procediment "*LS_SYS_PowerOff();"*, s'encarregarà de apagar el robot.

Explicació de les funcions secundaries:

Les funcions i els procediments secundaris són aquells que es fan servir al llarg del programa per implementar el funcionament del robot.

Funció "LS_NVOL_Read":

Aquesta funció permet llegir caràcter a caràcter els elements d'un fitxer que s'ha penjat a la memòria de l'LS Maker i s'ha obert prèviament.

```
LS_NVOL_Read(HANDLE,aux,1)
```

Procediment "LS_init":

La funció principal d'aquest procediment és el de posar en marxa l'LS Maker, i permetre així mateix la execució de les comandes inserides.

```
LS_Init();
```

Procediment "LS_Executiu":

Aquest procediment és essencial a l'hora d'implementar el codi necessari per el funcionament del LS_Maker. El LS_Executiu s'encarrega de forçar l'execució de les comandes, per el qual és molt important afegir-lo sempre que s'entri en un bucle, ja que aconseguirà executar la comanda.

```
LS_Executiu();
```

Funció "LS_NVOL_Open":

Aquesta funció és la responsable de la obertura del fitxer de text. Aquesta és equivalent a la funció "fopen" de la llibreria "stdio.h", però, en aquest cas, també s'encarrega de regular el port d'entrada del arxiu. A l'hora de penjar el fitxer de text al robot, cal assegurar-se de que s'escull el port A, donat que és aquest el que es farà servir per carregar l'arxiu al robot.

```
f = LS_NVOL_Open(NVOL_STREAM_A, "r");
```

Procediment "LS_NVOL_Close":

La funció principal de aquest procediment és el d'executar el tancament del fitxer de text obert al inici del programa.

```
LS_NVOL_Close(f);
```

Procediment "LS_LCD_Clear":

El procediment mencionat s'encarrega de netejar la pantalla del robot cada cop que s'ha pintat alguna cosa en ella.

```
LS_LCD_Clear();
```

Procediment "LS_LCD_GotoXY":

Col·loca el punter preparat per la escriptura a les coordenades 'x' i 'y' seleccionades de la pantalla led del robot.

```
LS_LCD_GotoXY(x,y);
```

Procediment "LS_LCD_Printf":

Com el seu nom indica, aquest procediment permet escriure a la pantalla led del LS_Maker.

```
LS_LCD_Printf(x,y,"%c",cadena[i]);
```

Procediment "LS_SYS_SleepMiliSecs":

La funcionalitat d'aquest procediment radica en el fet de provocar el "repòs" del robot durant un període limitat de segons escollits prèviament i expressats en milisegons. Durant aquest període de temps el robot romandrà quiet.

```
LS_SYS_SleepMiliSecs(2000);
```

Funció "LS_MT_Lineal":

Mitjançant aquesta funció s'aconsegueix que el LS_Maker es mogui cap endavant o bé cap endarrere segons la velocitat que se li passi. Si la velocitat que se li proporciona a la funció és negativa, el robot es mourà cap endarrere, mentre que, en cas contrari, es mourà cap endavant. Aquesta funció també depèn d'una variable de temps que se li proporcionarà arrel de la lectura del fitxer.

```
LS_MT_Lineal(temps, velocitat, 0, &stop);
```

Funció "LS_MT_GetTimeFromAngle":

Aquesta funció permet, com el seu nom indica, donades unes variables de velocitat i àngle de gir, extreure el temps que trigarà en realitzar la acció.

```
temps = LS_MT_GetTimeFromAngle(angle, velocitat);
```

Funció "LS_MT_TurnRight":

La funció "LS_MT_TurnRight" permet que el robot realitzi un gir cap a la dreta segons la velocitat i el temps que se li passin.

```
LS_MT_TurnRight(temps,velocitat,0,0,&stop);
```

Funció "LS_MT_TurnLeft":

La funcionalitat d'aquesta funció és essencialment la mateixa que la de la funció "LS_MT_TurnRight", però, en comptes de girar cap a la dreta girarà cap a l'esquerra.

```
LS_MT_TurnLeft(temps, velocitat,0,0,&stop);
```

Procediment "LS_SYS_PowerOff":

Com el seu nom indica, la funció bàsica d'aquest procediment és la de apagar el LS_Maker al final de l'execució de comandes.

```
LS_SYS_PowerOff();
```

Control d'errors :

A l'hora de realitzar aquesta fase de la pràctica, s'havien de realitzar certs controls per tal de que el robot efectués les comandes adequadament. El primer dels errors que calia comprovar era que, a l'hora de llegir les coordenades del missatge, aquestes entressin dins de l'interval permès, per tal de que es pintessin el missatge correctament per la pantalla del robot. És per això que, a l'hora d'implementar la primera part del procediment "LlegeixFiter" s`ha fet servir una variable error que, en cas de que les coordenades siguin correctes, permetrà l'execució de la resta de comandes. En cas contrari, no s'executaran les comandes llegides a la següent línia de codi, i saltarà directament a la següent comanda, després de pintar per pantalla un missatge d'error i romandre en espera durant cinc segons.

El segon cas que calia comprovar era, que les lletres que defineixen la comanda que el robot haurà d'executar fossin correctes. Per tal d'executar aquesta part, es va fer servir la comanda "*switch*", la qual executarà certes comandes segons la lletra. En cas de que la lletra sigues incorrecta, el "switch" accedia a la opció "default", en la qual es mostra un missatge d'error, i s'espera cinc segons fins a executar la següent comanda.

Un últim factor que també s'ha tingut en compte a l'hora de realitzar el codi, és que el temps d'execució del moviments del LS_Maker, proporcionat per el fitxer de text, sigues positiu i més gran a 0, donat que no es poden executar comandes en temps negatiu.

Fase 2

Descripció dels mòduls:

A l'hora d'implementar el codi de la pràctica, hem fet servir diversos mòduls, que per tal de poder separar els treball.

1.Mòdul "main":

Aquest mòdul és la base del projecte. El "main" només consta de un arxiu ".c", ja que no es poden implementar funcions i procediments en ell. En aquest mòdul generem un menú principal del programa, mitjançant el qual l'usuari podrà seleccionar les diferents opcions que composen el joc.

En primer lloc es generen les llistes que incloen la informació obtinguda arrel de la lectura del fitxers "Pilots.txt" i "Circuits.txt". Un cop s'ha pintat el menú d'opcions per pantalla, el qual conté un total de 9 opcions, segons la opció escollida per l'usuari s'executarà un procés o un altre.

Primerament a les opcions 1 i 2 se li ofereix la oportunitat a l'usuari d'afegir o eliminar circuits de la llista de pilots (lp).

La tercera i quarta opció, al igual que la primera i la segona, permeten a l'usuari afegir o eliminar circuits respectivament de la llista de circuits. En tot cas es modificarà la llista de circuits "lc".

La cinquena opció del menú mostra per pantalla tots els integrants de la llista de pilots i també mostra la seva informació com ara l'escuderia i la quantitat de punt que té. Per tal de realitzar-ho es truca a la opció "mostraPilots" implementada al mòdul.

A la sisena opció, el programa realitza les estadístiques de les curses i els campionats realitzats al llarg de l'execució del programa. Aquesta opció avalua els punts acumulats de cada pilot i la quantitat de vegades que s'ha corregut en un circuit.

La opció 7 del menú permet realitzar una cursa en un circuit escollit per l'usuari en la qual hi participaran tots els pilots disponibles. El resultat de la cursa es calcularà de forma aleatòria.

La opció 8 per el contrari permet a l'usuari realitzar un campionat, en el qual tots els integrants de la llista de pilots competiran en un nombre indefinit de curses a selecció de l'usuari. Els resultats del campionat modificaran la informació de les estadístiques dutes a terme a l'opció 6.

Per últim, la opció 9 és la opció de sortida del programa. Quan l'usuari entra en aquesta opció, s'actualitzen les dades del joc i es guarda la informació que es troba a les llistes de pilots i circuits novament en el fitxer de text amb el nom corresponent.

2.Mòdul "LlistaPDI":

L'objectiu principal d'aquest mòdul és la creació de les diferents llistes que farem servir al llarg de la pràctica i la implementació de les funcions i procediments de modificació de la cua. En aquest mòdul s'han implementat les mateixes funcions per a les dues llistes PDI generades, la de pilots i la de circuits. Com que les dues llistes funcionen de la mateixa manera, la explicació del les funcions i procediments es farà de forma genèrica en comptes de realitzar-la independentment per les dues llistes.

Explicació dels tipus:

1. **Pilots:** El tipus "Pilots" conté la informació referent als pilots amb els que es treballarà durant la pràctica. En aquest tipus podem trobar el nom del pilot representat amb una cadena de caràcters, el nom de la escuderia per a la que participa, també representada amb una cadena de caràcters. Per altra banda conté una variable enter amb la quantitat de punts acumulats per el pilot, y un real que conté la quantitat de kilòmetres que ha recorregut el pilot fins el moment. Per últim aquest tipus conté una llista PDI dintre seu, ja que aquesta es farà servir per a dur a terme la estadística durant el joc.

```
typedef struct {
    char nomPilot[MAX];
    char nomEscuderia[MAX];
    int punts;
    float kilometres;
    LlistaPDI_E le;
}Pilots;
```

2, **Estadistica:** El tipus "Estadistica" és el responsable d'emmagatzemar la informació dels resultats obtinguts als diferents circuits per a cada jugador. Aquest contindrà el nom del pilot, la seva el nom del circuit en el que acaba de participar, i la quantitat de punts obtinguda en aquell circuit.

```
17 typedef struct {
18     char nomPilot[MAX];
19     char nomCircuit[MAX];
20     int punts;
21 }Estadistica;
```

3. **Circuits:** El tipus "Circuits" conté tota la informació referent als circuits amb els que es treballaran al llarg de la pràctica. En aquest tipus s'emmagatzemen el nom del circuit en format de cadena de caràcters,, un real amb els kilòmetres de llargada del circuit, i un enter amb la quantitat de punts màxima que es pot aconseguir en aquell circuit.

```
23 typedef struct {
24     char nomCircuit[MAX];
25     float kilometres;
26     int PuntsMax;
27 }Circuits;
```

4. **NodeP:**El tipus "NodeP" és el tipus principal que conforma la "LLISTAPDI_P". Aquest "NodeP" conté un element tipus "Pilot" i un punter que apunta a un altre element tipus "NodeP".

```
29 typedef struct Np{
30     Pilots p;
31     struct Np* seg;
32 }NodeP;
```

5. **NodeE:** El tipus "NodeE" és el tipus principal que conforma la "LLISTAPDI_E". Aquest "NodeE" conté un element tipus "Estadisticat" i un punter que apunta a un altre element tipus "NodeE".

```
34 typedef struct Ne{
35     Estadistica e;
36     struct Ne* seg;
37 }NodeE;
```

6. **NodeC:** Tal i com s'ha mencionat en el cas del "NodeP", el "NodeC" és la base de la llista de circuits "lc" ja que conté un element tipus "Circuits" on estarà emmagatzemada la informació d'un dels circuits, i un punter que apunta a un altre element "NodeC".

```
39 typedef struct Nc{
40     Circuits c;
41     struct Nc* seg;
42 }NodeC;
```

7. **LLISTAPDI_P:** Aquest tipus genera una llista PDI de pilots, ja que està compost per dos "NodeP", un que representarà el primer element de la llista, el "fantasma", anomenat "pri", i un segon que marcarà sempre l'element que va abans del PDI, anomenat "ant".

```
44 typedef struct{
45     NodeP* pri;
46     NodeP* ant;
47 }LlistaPDI_P;
48
```

8. **LLISTAPDI_E:** Aquest tipus genera una llista PDI de "Estadistica", ja que està compost per dos "NodeE", un que representarà el primer element de la llista, el "fantasma", anomenat "pri", i un segon que marcarà sempre l'element que va abans del PDI, anomenat "ant".

```
49 typedef struct{
50     NodeE* pri;
51     NodeE* ant;
52 }LlistaPDI_E;
```

9. **LLISTAPDI_C:** Aquest tipus és el tipus generador de la llista PDI de circuits, ja que conté dos elements "NodeC" que representaran el primer element, el "fantasma", anomenat "pri", i l'element anterior al PDI, anomenat "ant".

```
54 typedef struct{
55     NodeC* pri;
56     NodeC* ant;
57 }LlistaPDI_C;
```

```
LlistaPDI_P LLISTAPDI_P_crea ();
LlistaPDI_P LLISTAPDI_P_insereix (LlistaPDI_P  lp, Pilots p);
LlistaPDI_P LLISTAPDI_P_esborra (LlistaPDI_P lp);
Pilots  LLISTAPDI_P_consulta (LlistaPDI_P lp);
int LLISTAPDI_P_buida (LlistaPDI_P lp);
LlistaPDI_P LLISTAPDI_P_vesInici (LlistaPDI_P lp);
LlistaPDI_P LLISTAPDI_P_avanca (LlistaPDI_P lp);
int LLISTAPDI_P_fi (LlistaPDI_P lp);
LlistaPDI_P LLISTAPDI_P_destrueix (LlistaPDI_P lp);

LlistaPDI_C LLISTAPDI_C_crea ();
LlistaPDI_C LLISTAPDI_C_insereix (LlistaPDI_C lc, Circuits c);
LlistaPDI_C LLISTAPDI_C_esborra (LlistaPDI_C lc);
Circuits LLISTAPDI_C_consulta (LlistaPDI_C lc);
int LLISTAPDI_C_buida (LlistaPDI_C lc);
LlistaPDI_C LLISTAPDI_C_vesInici (LlistaPDI_C lc);
LlistaPDI_C LLISTAPDI_C_avanca (LlistaPDI_C lc);
int LLISTAPDI_C_fi (LlistaPDI_C lc);
LlistaPDI_C LLISTAPDI_C_destrueix (LlistaPDI_C lc);
```

```
ListaPDI_E LLISTAPDI_E_crea();
ListaPDI_E LLISTAPDI_E_insereix(ListaPDI_E le, Estadistica e);
ListaPDI_E LLISTAPDI_E_esborra(ListaPDI_E le);
Estadistica LLISTAPDI_E_consulta(ListaPDI_E le);
int LLISTAPDI_E_buida(ListaPDI_E le);
ListaPDI_E LLISTAPDI_E_vesInici(ListaPDI_E le);
ListaPDI_E LLISTAPDI_E_avanca(ListaPDI_E le);
int LLISTAPDI_E_fi(ListaPDI_E le);
ListaPDI_E LLISTAPDI_E_destrueix(ListaPDI_E le);

#endif
```

1. Funció: "LLISTAPDI_crea"

La funció "LLISTAPDI_crea", és l'encarregada de generar les variables de tipus "LlistaPDI". Primerament es genera una variable tipus llista i una variable del tipus de element que s'hi vol inserir. Un cop generats, es demana memòria per generar el primer element de la llista. En cas de que no hi hagi memòria disponible, la funció retornarà un "NULL", és a dir, un punter que no apunta a res. En cas contrari, es guardarà l'element desitjat i es generarà un punter que, més endavant apuntarà al següent element, però, mentrestant apuntarà a "NULL".

```
ListaPDI_P LLISTAPDI_P_crea ();
```

2. Funció: "LLISTAPDI_insereix"

Aquesta funció és la encarregada de afegir-li elements a la llista. A aquesta llista se li passen tant la variable de "LListaPDI" i l'element tipus assignat segons la llista, el qual se li vol afegir. La informació de l'element es guardarà en aquella posició on estigui apuntant el "PDI", i per últim retornarà la llista amb el nou element inserit.

```
ListaPDI_P LLISTAPDI_P_insereix (ListaPDI_P  lp, Pilots p);
```

3. Funció: "LLISTAPDI_esborra"

La tercera funció d'aquest mòdul és la encarregada de esborrar elements de la llista. Quan l'usuari truca a aquesta funció, li passa la variable tipus "LlistaPDI", la qual està té el PDI apuntant a l'element que es vol eliminar. En cas de que el PDI apunti a "NULL", significarà que la llista estarà buida, i per tant, no es podrà eliminar cap element més. En canvi, si el PDI no apunta a "NULL", es truca a un element auxiliar que facilitarà el traspàs de nodes de l'element anterior al PDI al següent, i tot seguit s'alliberarà el node que apunta a l'element desitjat, eliminant-lo de la llista. El PDI s'haurà mogut al següent element de la llista.

```
ListaPDI_P LLISTAPDI_P_esborra (ListaPDI_P lp);
```

4. Funció: "LLISTAPDI_consulta"

Mitjançant aquesta funció l'usuari pot consultar la informació que conté l'element on està situat el PDI. Aquesta funció retorna el valor de l'element a consultar, però, en cas de que la llista estigui buida, mostrarà un missatge d'error, i retornarà un valor de l'element aleatori.

```
Pilots  LLISTAPDI_P_consulta (LlistaPDI_P lp);
```

El funcionament d'aquesta funció és molt senzilla. Donat que lo que es vol retornar és un valor booleà que indiqui si la llista encara té elements o no, es retorna el resultat de comparar la situació en la que el primer element de la llista apunti a "NULL". Si aquesta comparació retorna un '1', significa que es compleix i per tant estarà buida. En canvi, si retorna un '0', significa que no es compleix i encarà en queden elements a la llista.

```
int LLISTAPDI_P_buida (LlistaPDI_P lp);
```

6. Funció: "LLISTAPDI_vesInici"

Com el seu nom indica, la funcionalitat d'aquesta funció recau en el fet de situar el PDI al primer element de la llista. Per poder fer-ho, es fa que el punter que apunta a l'element anterior al PDI, sigui quina sigui la seva posició, apunti al primer element de la llista, el "fantasma", el quant no conta a l'hora de fer l'execució de la llista. Per tant, el PDI es situarà al primer element. Un cop fet es retornarà la llista.

```
LlistaPDI_P LLISTAPDI_P_vesInici (LlistaPDI_P lp);
```

7. Funció: "LLISTAPDI_avanca"

La setena funció té com objectiu avançar el PDI un element, per tal de que es mogui a l'element següent. El seu funcionament és simple, el PDI es mourà a l'element on està apuntant l'element on es troba en aquell moment, provocant així el seu desplaçament al següent element. Un cop realitzat retornarà de nou la llista modificada.

```
LlistaPDI_P LLISTAPDI_P_avanca (LlistaPDI_P lp);
```

8. Funció: "LLISTAPDI_fi":

Aquesta funció, de forma semblant a la funció "LLISTAPDI_buida" retorna un valor d'enter segons si es dona la situació en la que el PDI està apuntant a "NULL". Si aquest

és el cas, retornarà un '1', indicant que, efectivament, el PDI es troba a l'últim element. En cas contrari, retornarà un '0'.

```
int LLISTAPDI_P_fi (LlistaPDI_P lp);
```

9. Funció: "LLISTAPDI_destrueix":

La funció "LLISTAPDI_destrueix", com el seu nom indica, és l'encarregada de buidar la llista d'elements. Per tal de aconseguir-ho primer es genera un punter auxiliar del mateix tipus que el dels integrants de la llista. Aquest element es situa primerament en el primer element de la llista. Un cop situat avança el PDI en una casella i després, s'allibera l'auxiliar , eliminant-lo. Per últim l'auxiliar apunta a la nova posició del PDI. Aquest procés és repeteix fins que només queda un element a la llista, el qual és eliminat.

```
LlistaPDI_P LLISTAPDI_P_destrueix (LlistaPDI_P lp);
```

3.Mòdul "Logica":

Aquest mòdul, tal i com el seu nom indica, és l'encarregat del funcionament de la lògica del circuit. Totes les funcions que no es dediquen a la implementació de les diferents llistes, són generades en aquest mòdul. Aquest consta d'un total de 20 procediments i funcions que implementen el funcionament sencer de aquesta fase.

```
#ifndef _LOGICA_H_
#define _LOGICA_H_
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include "LlistaPDI.h"
#include "Cua.h"

LlistaPDI_P extreuInfo(LlistaPDI_P lp, FILE *f);

LlistaPDI_C extreuCircuits(LlistaPDI_C lc, FILE *f_c);

LlistaPDI_P afegeixJugador(LlistaPDI_P lp, char nom_pilot[MAX], char nom_escuderia[MAX]);

LlistaPDI_P esborraPilot(LlistaPDI_P lp, char nom_pilot[MAX]);

LlistaPDI_C afegeixCircuit(LlistaPDI_C lc, char nom_cursa[MAX], float km, int punts);

LlistaPDI_C esborraCircuit(LlistaPDI_C lc, char nom_circuit[MAX]);

void mostraPilots(LlistaPDI_P lp);

void circuitsDisponibles(LlistaPDI_C lc);

LlistaPDI_E circuitEscollit(LlistaPDI_C lc, char nom_circuit[MAX], LlistaPDI_P lp, LlistaPDI_E le);

LlistaPDI_E ferCursa(Circuits c, LlistaPDI_P lp, LlistaPDI_E le);

int PuntsMaximsPilots(LlistaPDI_P lp);

float KmMaximsPilots(LlistaPDI_P lp);

Cua circuitAuxiliar(char nom_circuit[MAX], Cua c, LlistaPDI_C lc);

LlistaPDI_E ferCampionat(Cua c, LlistaPDI_P lp, LlistaPDI_E le);

LlistaPDI_P insereixPilot(LlistaPDI_P lp, FILE *f);

LlistaPDI_C insereixCircuit(LlistaPDI_C lc, FILE *f_c);

void printaEstadistiques(LlistaPDI_E le, LlistaPDI_C lc);
#endif
```

1. Funció: "LLISTAPDI_P extreuInfo"

Aquesta funció és l'encarregada de emmagatzemar la informació que es troba en el fitxer "Pilots.txt". Primerament genera una llista de pilots mitjançant la funció "LLISTAPDI_P_crea". Seguidament es va emmagatzemant la informació que es va llegint del fitxer en unes variables auxiliars. Aquestes variables auxiliars permeten emmagatzemar la informació llegida en els diversos paràmetres del tipus "Pilot".
Primerament s'extreuen el nom del pilot i el de l'escuderia. Per altra banda es converteixen els elements numèrics de la llista en números. La puntuació es converteix en un enter mitjançant la comanda "atoi", implementada a la llibreria "stdlib.h", i per altra banda es transformen els kilòmetres en un real, amb la funció "atof", també de la mateixa llibreria que l'anterior.

Un cop obtinguts tots els elements, s'insereixen en els seus respectius camps a la variable tipus "Pilot", i per últim, es retorna la llista "lp" creada.

```
LlistaPDI_P extreuInfo(LlistaPDI_P lp, FILE *f, int *sortir);
```

2. Funció: "extreuCircuits":

El funcionament d'aquest procediment té la mateixa base que el procediment anterior, i és, emmagatzemar les dades del fitxer "Circuits.txt" en una llista per així fer-la servir al llarg de la pràctica. Primerament en aquesta funció és genera una llista mitjançant la comanda "LLISTAPDI_C_crea", i se li aniran inserint els elements mitjançant la comanda "LLISTAPDI_C_insereix". Al igual que en la darrera funció, s'han creat diverses variables auxiliars, en les quals anirem emmagatzemant la informació que es va llegint del fitxer. Un cop llegits el nom del circuit, els seus kilòmetres i la quantitat màxima de punts que es poden obtenir, aquestes variables s'introdueixen en una altre variable de tipus "Circuits", i s'insereix a la llista com un nou element.

Per últim aquesta funció retorna la variable de la nova llista generada i que conté els elements del fitxer.

```
LlistaPDI_C extreuCircuits(LlistaPDI_C lc, FILE *f_c, int *sortir);
```

3. Funció: "afegeixJugador":

Aquesta funció permet a l'usuari afegir un nou pilot a la llista "lp", la qual emmagatzema la informació de tots els pilots fins el moment. Mentre el programa va guardant la informació inserida per l'usuari, es va comprovant que aquesta sigui correcta, i, que el pilot que es vulgui afegir no existeixi a la llista. En cas de que un dels casos anteriors es compleixi, la funció mostrarà un missatge d'error i tornarà al menú principal, retornant el valor de la llista sense variar. Un cop ja s'hagi guardat la informació correcta del pilot a llista, la funció retornarà el seu valor.

```
LlistaPDI_P afegeixJugador(LlistaPDI_P lp, char nom_pilot[MAX], char nom_escuderia[MAX]);
```

4. Funció: "esborraPilot":

Al trucar aquesta funció des-del menú principal del joc, se li passen les variables que contenen la informació del pilot que l'usuari vol eliminar i la variable de la llista de la qual es vol eliminar. Un cop s'ha accedit a la funció, aquesta comprova que el pilot que l'usuari vol esborrar existeixi dins de la llista "lp". En cas de que no existeixi, mostrarà un missatge d'error avisant de que aquest no existeix, i retornarà el valor de la llista "lp" sense modificar. En cas contrari, es recorrerà la llista en busca del pilot, comparant si també està el seu nom escrit amb majúscules o minúscules, i, un cop el trobi, cridarà a la funció "LLISTAPDI_P_esborra", implementada al mòdul anterior, per tal d'eliminar el pilot. Un cop executat, la funció retorna el valor de la llista modificada.

```
LlistaPDI_P esborraPilot(LlistaPDI_P lp, char nom_pilot[MAX]);
```

5. Funció: "afegeixCircuit":

Mitjançant aquesta funció, l'usuari podrà afegir un circuit nou a la llista "lc" de tipus "LLISTAPDI_C". La funció rep les variables que contenen la informació del circuit que es vol inserir en un element de tipus "Circuits" i la llista "lc" on es vol inserir el circuit. Abans de afegir aquest element a la llista, es comprova que la informació inserida sigui correcta i que el circuit es vol inserir no existeixi a la llista de circuits. En cas de que un dels errors anomenats anteriorment es doni, es mostrarà un missatge d'error i tornarà al menú principal, retornant el valor de la llista sense modificar. En cas contrari, s'afegirà el nou circuit mitjançant la funció "LLISTAPDI_C_insereix" i es retornarà el valor de la llista modificada.

```
LlistaPDI_C afegeixCircuit(LlistaPDI_C lc, char nom_cursa[MAX], float km, int punts);
```

6. funció: "esborraCircuit":

La funció "esborraCircuit" és la encarregada d'eliminar un circuit escollit per l'usuari. Primerament es genera una variable tipus "Circuit" que contindrà la informació del circuit proporcionada per l'usuari. Abans de intentar eliminar el circuit, la funció s'encarrega de comprovar que el circuit que es vol eliminar existeixi a la llista, ja que, en cas contrari, podria provocar un "*core dumped*". Si es dona la situació en la que circuit no existeixi, la funció retornarà un missatge d'error i tornarà al menú principal, retornant el valor de la llista sense modificar. En cas contrari, mitjançant la funció "LLISTAPDI_C_esborra" s'esborrarà el circuit desitjat. Per últim la funció retornarà el valor de la llista modificada.

```
LlistaPDI_C esborraCircuit(LlistaPDI_C lc, char nom_circuit[MAX]);
```

7. Procediment: "mostraPilots":

Com el seu nom indica, la funció d'aquest procediment és el de mostrar tots els pilots disponibles per pantalla, incloent-hi el seu nom, la seva escuderia , els kilòmetres màxims que ha corregut, i la quantitat total de punts que té. Per tal de aconseguir-ho genera un bucle que, des-de el primer element de la llista de pilots "lp", va mostrant per pantalla la informació de tots els elements de la llista sencera, és a dir, tots els pilots.

```
void mostraPilots(LlistaPDI_P lp);
```

8. Procediment: "circuitsDisponibles":

Donat que la funcionalitat d'aquest procediment i l'anterior són iguals, la implementació d'aquest és similar a l'anterior, però, en comptes de treballar amb la llista de pilots "lp" es treballa amb la llista de circuits "lc". Al igual que al procediment anterior es genera un bucle que va mostrant per pantalla tots els elements de la llista des-de el primer fins l'últim.

```
void circuitsDisponibles(LlistaPDI_C lc, int *mal);
```

9.Procediment: "circuitEscollit":

Aquest procediment representa una part de la comanda "ferCursa" , la qual serveix per realitzar una única cursa amb tots els pilots dels que es disposen. En aquest procediment es reben com a paràmetres la llista de circuits, la llista de pilots, la llista que contindrà les dades de la estadística, i el nom del circuit en el qual l'usuari vol realitzar la cursa. Un cop tenim emmagatzemat el nom del circuit, es situa el PDI de la llista de curses en la primera posició. Al llarg del procediment es va comparant el nom del circuit amb cadascun dels circuits que es troben a la llista. En cas de que coincideixin es mostrarà per pantalla el nom del circuit que s'ha decidit buscar i s'executarà la funció "ferCursa". En cas de que no es trobo el circuit, mostrarà el missatge d'error.

```
LlistaPDI_P circuitEscollit(LlistaPDI_C lc, char nom_circuit[MAX], LlistaPDI_P lp);
```

10. Funció: "ferCursa":

Aquesta funció és l'encarregada de realitzar les curses del joca la que se li passen la llista de circuits, la de pilots i la de estadístiques. Primerament es genera una variable tipus "pilot", on s'emmagatzemarà la informació del pilot seleccionat pel "PDI", i una altre variable tipus enter anomenada "Puntuació", la qual es modificarà en funció de la puntuació del pilot que s'ha consultat. En cas de que aquesta sigui '0' se li assignarà un valor aleatori i, en cas contrari, se li assignarà una puntuació segons la formula

proporcionada a l'enunciat. Un cop finalitzada l'acció es retorna la llista amb els pilots modificada.

```
ListaPDI_P ferCursa(Circuits c, ListaPDI_P lp);
```

12. Funció: "LLISTAPDI_P_substitueix":

Aquesta funció s'encarrega de substituir l'element seleccionat pel "PDI" per un de nou. Aquesta funció es fa servir a la funció "FerCursa".

```
ListaPDI_P substitueix(ListaPDI_P lp, Pilots p){
```

13. Funció: "LLISTAPDI_P_insereixEstadistica":

Aquesta funció ens permet inserir els diferents circuits on ha participat el jugador. En cas de que el jugador ja hagi participat anteriorment en aquell circuit es comparará la puntuació obtinguda i es quedarà amb la puntuació més alta obtinguda en aquell circuit.

```
ListaPDI_E insereixEstadistica(ListaPDI_E le, Estadistica e){
```

14. Funció: "PuntsMaximsPilots":

Aquesta funció rep com a paràmetre la llista de pilots, ja que el seu objectiu és sumar la quantitat total de punts de tots els pilots, i retornar-la.

```
int PuntsMaximsPilots(ListaPDI_P lp)
```

15. Funció: "KmMaxPilots":

Aquesta funció rep com a paràmetres la llista de pilots, i retorna un real amb la suma total de kilòmetres correguts per tots els pilots.

```
float KmMaximsPilots (ListaPDI_P lp)
```

16. Funció: "circuitAuxiliar":

Aquesta funció rep com a paràmetres la llista de circuits, la cua i el nom d'un circuit. Primerament es generen dues llistes auxiliars en que, farem servir per localitzar el circuit seleccionat per l'usuari dins de la llista de circuits. A una d'elles guardarem les dades inserides per l'usuari i a la segona emmagatzemarem les dades llegides del elements de la llista.

```
Cua circuitAuxiliar (char nom_circuit[MAX], Cua c, LlistaPDI_C lc)
```

17. Funció: "ferCampionat":

Aquesta funció té com a objectiu principal realitzar el campionat del joc, per aquest motiu se li passen les variables c, lp, que contenen la informació dels circuits, escollits en ordre per l'usuari, la llista de pilots. Aquesta funció crida a la funció "ferCursa",per a cada circuit indicat per l'usuari. Per últim s'emmagatzema tota la informació de nou en les llistes corresponents. Aquest procés es repetirà fins que la cua de circuits escollits per l'usuari es buidi.

```
LlistaPDI_P ferCampionat(Cua c, LlistaPDI_P lp);
```

18. Funció: "insereixPilot":

La funció "insereixPilots" serveix per tornar a emmagatzemar la llista de pilots dins el fitxer de text d'on els vam llegir originàriament. Primerament ens guardem la informació de cada pilot en variables auxiliars, i, un cop les tenim, escrivim sobre el fitxer la informació en una sola línia, i avancem al següent element. Un cop realitzat tot el procés, eliminem la llista.

```
LlistaPDI_P insereixPilot(LlistaPDI_P lp, FILE *f)
```

19. Funció: "insereixCircuit":

El funcionament d'aquesta funció és molt semblant a l'anterior, però en aquest cas, guardarem la informació de tots els fitxers dins del fitxer de text que li pertoca. Com a la funció anterior, realitzem un bucle que emmagatzema la informació de cada circuit i l'escriu al fitxer. Un cop està tota la informació emmagatzemada, s'elimina la llista.

```
LlistaPDI_C insereixCircuit(LlistaPDI_C lc, FILE *f_c)
```

20. Procediment: "printaEstadístiques":

Aquest procediment té com a objectiu mostrar les estadístiques amb els punts màxims acumulats de cada pilot per a cada circuit. Aquest procediment es realitzarà constantment, fins que s'hagin mostrar les puntuacions de tots els circuits on han tingut lloc les curses del campionat.

```
void printaEstadistiques(LlistaPDI_P lp, LlistaPDI_C lc);
```

21. Procediment: “malluscules”:

Aquest procediment té una funcionalitat molt bàsica: aquest procediment té com a objectiu transformar totes les lletres d’una cadena de caràcters en majúscules, per tal de que així es puguin fer servir en diversos processos de control d’errors.

```
void malluscules(char nom[MAX]){
```

4.Mòdul “Cua”:

El mòdul “Cua” és un mòdul senzill que serveix per a implementar una cua dinàmica on es emmagatzemaran els circuits que participaran al campionat.

```
#ifndef _CUA_H_
#define _CUA_H_

#define MAX 100

#include <stdlib.h>
#include <stdio.h>

typedef struct {
    char nomCircuit[MAX];
    float Kilometres;
    int punts;
}CircuitsCua;

typedef struct N {
    CircuitsCua C;
    struct N*seg;
}NodeCua;

typedef struct{
    NodeCua *pri;
    NodeCua *ult;
}Cua;

Cua CUA_crea();
void CUA_encua(Cua *c, CircuitsCua C);
void CUA_desencua(Cua *c);
CircuitsCua CUA_cap(Cua c);
int CUA_buida(Cua c);
void CUA_destueix(Cua *c);

#endif
```

Explicació dels tipus:

1. **CircuitsCua:** Aquest tipus es el que ens servirà de referència a l'hora d'implementar la cua dinàmica de circuits del campionat. Aquest tipus conté bàsicament els mateixos elements que el tipus "Circuits" implementats al mòdul "LLISTAPDI". Aquest tipus ens servirà per emmagatzemar les dades de cadascun dels circuits.

```
 9 typedef struct {
10     char nomCircuit[MAX];
11     float Kilometres;
12     int punts;
13 }CircuitsCua;
```

2. **NodeCua:** En aquest cas es genera un tipus que es farà servir per implementar la cua. El "NodeCua" conté un element de tipus "CircuitsCua" i un punter al propi tipus, ja que es farà servir a la llista per apuntar a els altres elements.

```
15 typedef struct N {
16     CircuitsCua C;
17     struct N*seg;
18 }NodeCua;
```

3. **Cua:** Aquest tipus és el que es definirà com la cua dinàmica. Aquest conté dos punters , un anomenat "pri" que representarà el primer element de la cua, i un altre anomenat "ult", que representarà l'últim element de la llista. Tots dos nodes són "NodeCua" per tant contenen la informació emmagatzemada a variables tipus "CircuitsCua".

```
20 typedef struct{
21     NodeCua *pri;
22     NodeCua *ult;
23 }Cua;
```

1.Funcio: "CUA_crea":

La funció "CUA_crea" genera una variable tipus "Cua" i es demana espai de memòria per a la seva generació. En cas de que no hi hagi espai disponible, retornarà un missatge d'error i el valor "NULL" per la variable de sortida. En cas de que si que hi hagi espai, el punter de "pri" apuntarà a la Cua, i es col·loca el punter de l'últim apuntant al primer. Un cop realitzat el procés, retornarà el valor de la cua generada.

```
25 Cua CUA_crea();
```

2.Procediment: "CUA_encua":

Aquest procediment és l'encarregat d'afegir nous elements de tipus "NodeCua" a la nostra cua. En primer lloc aquests nous elements s'afegeixen en l'última posició, ja que el funcionament bàsic de la cua és pot comparar amb l'estructura FIFO (first in first out). Un col·locat el nou element a la llista, el punter "ult" es mou sobre el nou element inserit.

```
void CUA_encua(Cua *c, CircuitsCua C)
```

3.Procediment: "CUA_desencua":

En aquets cas, contràriament al cas anterior, l'objectiu de procediment és el de eliminar el primer element que es troba a la llista, és a dir, aquell on apunta el punter "pri", ja que es tracta de una cua. Per això creem una variable auxiliar, que farem servir per, en primer lloc, apuntat al primer element i així poder moure el punter "pri" al següent element de la llista, abans d'eliminar-lo.

```
void CUA_desencua(Cua *c)
```

4. Funció : "CUA_cap":

Aquesta funció té com a objectiu retornar el valor de l'element situat a la primera posició de la cua. En cas de que la cua sigui buida pintarà per pantalla un missatge d'error. En cas contrari retornarà l'element mencionat.

```
CircuitsCua CUA_cap(Cua c)
```

5. Funció : "CUA_buida":

Aquesta funció s'encarrega de mostrar l'estat de la cua, és a dir, si aquesta es troba buida o no. En cas de que ho estigui retornarà un '1', en canvi, si encara en queden elements, retornarà un '0'.

```
int CUA_buida(Cua c)
```

6. Procediment : "CUA_destrueix":

Aquest procediment, com el seu nom indica, és l'encarregat de destruir la cua. Per això, primerament es genera una variable auxiliar, mitjançant la qual anirem eliminem tots els elements de la cua fins que no en quedin. A més a més, col·loquem el punter que apunta a l'últim element també a "NULL", ja que la cadena estarà buida.

```
void CUA_destrueix(Cua *c)
```

Descripció dels "TADs":

A continuació es realitzarà una breu explicació dels TAD's (estructures) empleats en aquest programa.

En primer lloc hem fet servir una llista "PDI" anomenada "LLISTAPDI_P", mitjançant la qual hem pogut emmagatzemar la informació dels pilots que, per una banda obteníem del seu fitxer de text corresponent, i, per altra banda, afegia l'usuari a partir del menú principal del programa. Aquesta llista era imprescindible, ja que ens ha permès maniobrar amb les dades inserides per l'usuari i així mateix realitzar canvis de la seva informació cada cop que es realitzava una cursa.

Per altra banda també contàvem amb una altra llista "PDI" anomenada "LLISTAPDI_C", que, en aquest cas, contenia tota la informació de tots els circuits disponibles per a competir. Aquesta serveix per, gràcies a la disponibilitat dels diferents circuits, poder destacar diferents fases al llarg del joc.

A continuació, també cal destacar la utilització d'una tercera llista "PDI" anomenada "LLISTAPDI_E", la qual hem fet servir per tal de dur a terme les estadístiques del joc. A cada casella de la llista torbarem un element de tipus "Estadistica", i un punter cap al següent element, els quals es veuran modificats cada cop que s'executi una carrera. Aquesta llista, a més a més, es troba dins de la llista "LLISTAPDI_P", ja que per a cada pilot s'haurà de guardar la millor puntuació aconseguida per a cada circuit que ha participat.

Per últim he, fet servir una estructura de tipus cua anomenada "CUA", la qual s'ha fet servir, per emmagatzemar de forma organitzada, ja que es tracta d'una estructura de caràcter FIFO, els circuits que l'usuari vol que hi participin en el campionat. Aquesta cua s'anirà omplint cada cop que l'usuari escull un circuit, i es buidarà un cop s'hagi realitzat el campionat.

3.Problemes Observats

Fase 1

A l'hora d'implementar la fase 1 de la pràctica, amb la qual treballàvem directament amb el LS Maker, ens hem trobat diversos problemes que ens han portat a reescriure el codi diverses vegades. El primer problema amb el que ens vam trobar va ser, que a l'hora d'executar el codi amb el LS Maker, aquest es quedava en un bucle al pintat "Preparats!" per la pantalla. Aquest problema es va donar principalment, perquè desconeixíem la funcionalitat de la comanda "LS_Executiu();", la qual permet el funcionament correcte de les comandes. Donat a que no l'havíem fet servir, les comandes utilitzades en tots els bucles no s'executaven, i per tant, no entrava mai en la execució de la comanda.

Després de un replantejament del codi ens vam adonar, que, part del motiu de que no s'executessin les comandes es devia a que, tant la funció "atoi" com la funció "LS_NVOL_ReadLine();" estaven implementades incorrectament.

Malauradament aquest fet no va resoldre el problema i vam tornar a reescriure el codi. Tot i així, quan vam reescriure el codi ens vam trobar amb un problema que no sabíem resoldre. Quan el LS Maker executava les comandes, no executava aquelles que el feien moure's cap a la esquerra. Encara que el codi sigues exacte al del moviment cap a la dreta, canviant-li només la operació: "LS_MT_TurnRight" per "LS_MT_TurnLeft"; el moviment no s'executava, i es mantenia durant el temps d'execució quiet. Passat aquest interval de temps, el LS_Maker executava la resta de comandes.

Per poder solucionar aquest problema, hem tingut que reescriure el codi sencer en un altre ordinador, ja que en el ordinador on s'havia escrit originalment, es donaven errors provocats pel propi sistema. Un cop vam reescriure el codi, replantejant la extracció de la informació del arxiu i la seva interpretació, vam aconseguir que, finalment, l'LS_Maker es mogués en totes les direccions.

Fase 2

Durant la fase 2 d'aquesta pràctica ens hem anat trobant amb problemes molt diversos. Per començar a l'hora de executar el programa vam tenir un problema amb el "makefile" ja que no compilava i sortien errors que no coneixíem. Aquests errors es devien a que a l'hora de implementar el "makefile" (aquest fitxer es el que uneix tots i fa de compilador, si no estan ben escrits o implementats poden causar problemes desconeguts per a nosaltres) hi havia massa separació entre línia i línia, i això provocava que no funciones.

Més endavant ens vam trobar que moltes de les implementacions que fèiem de les llistes o cues no funcionaven pel fet d'errors minúsculs. Aquest problema ha sigut un el que hem anat arrossegant des de inicis del curs amb el terminal vela. La solució la vam trobar buscant per diferents pagines web on se'ns mostraven diferents comandes per a una terminal. La que nosaltres buscàvem més específicament era: "vsplit". Aquesta comanda et dona la possibilitat d'obrir un nou fitxer partint la pantalla per la meitat o segons el nombre de fitxer que tinguis oberts.

En alguns problemes no posaven específicament quin error feia el que ho provocava. Podien haver diferents maneres de solucionar-ho. Una opció era que haguessis tancat malament i et faltes un "}" en algun lloc. Un altre era que l'error estigues mes a dalt i provoques que es solucionessin altres de mes a baix.

El problema que ens ha portat més treball era el fet que les opcions "FerCursa", "FerCampionat" i "Estadistica", no acabaven de funcionar, i, com que aquestes tres depenen les unes de les altres, no acabàvem de arribar a la solució del problema. A l'hora d'implementar "FerCursa" , la comanda "rand()" ens donava sempre "0.00" , ja que la fèiem servir en funció del temps i no ho acabàvem de dominar. Aquest fet també ens provocava problemes a l'hora de realitzar el càlcul de la puntuació, ja que, com que la puntuació es multiplica per la sort, la qual es generava de forma aleatòria, el resultat sempre donava 0.

Per altra banda, també vam tindre problemes a l'hora d'emmagatzemar i actualitzar la informació dels pilots a les diferents llistes, ja que per dur-ho a terme cal desplaçar-se dins d'aquestes i, amb molta probabilitat, ens torbàvem amb masses salts de element. Aquests salts addicionals provocaven moltes vegades errors els quals ens van ser molt difícils de detectar.

Per últim, el problema més gran que vam tindre va ser que, a l'hora de programar l'estadística, no vam crear una llista addicional dintre de la llista de pilots, per el que, ens van sorgir molts errors que provocaven bucles infinits a l'execució del programa i, que, a més a més, provocaven que la opció de "FerCampionat" no funcionés correctament.

4.Conclusions

Fase 1

La fase 1 d'aquesta pràctica ens ha permès poder aplicar els nostres coneixements de programació en un àmbit més "real" de el que l'havíem fet servir fins el moment. El fet de tindre que treballar amb elements externs com ara el LSMaker, ens ha ajudat a comprendre millor que la programació no només té influència a nivell "gràfic". És a dir, no només serveix per dur a terme programes que es visualitzen a partir de la pantalla de l'ordinador, sinó que també té molta influència a nivell de components. Part d'aquesta influència va quedar demostrada a la segona pràctica de I.O., on el funcionament del circuit que havíem d'implementar depenia en un 80% del programa que li havíem de pujar a la seva memòria.

Per altra banda, aquesta fase ens ha servit com a entrenament de la utilització de funcions i procediments, ja que, per poder dur-lo a terme, calia, no només implementar funcions i procediments propis que ens ajudessin a facilitar la tasca, sinó també comprendre el funcionament de les diferents funcions i procediments proporcionats a classe. Aquests procediments i funcions eren essencials, ja que sense ells no tindríem cap influència en LSMaker, donat que aquestes són específiques per el seu funcionament.

Tanmateix però, ens hem adonat que cal replantejar - ser molt la funció del programa en comptes d'escriure el codi directament. Aquest fet es el que ens va portar a que, quan vam fer la pràctica el primer cop, l'LSMaker no es mogués cap a l'esquerra. Aquest fet ens va fer perdre molt de temps, encara que la pràctica no fos complicada. Degut al fet de precipitar-nos, vam tindre que reescriure el codi diverses vegades abans de trobar la solució al nostre problema. Encara que, ni tan sols ara comprenem del tot per què l'LSMaker no es movia cap a l'esquerra.

Fase 2

Al contrari que a la fase 1 d'aquesta pràctica, en la fase 2 ens hem trobat amb molts problemes degut a l'extensió d'aquesta. Tot i que la quantitat de procediments i funcions sigues inferior a la de l'anterior pràctica, el codi que es trobava a cadascun d'ells era molt més extens, motiu que ens va provocar una gran quantitat de problemes tal i com hem esmentat anteriorment.

Donat que aquesta és la primera pràctica amb la que hem tingut que treballar amb estructures dinàmiques, el plantejament d'aquesta ha sigut molt divers del-de la resta de pràctiques realitzades al llarg del curs, ja que per poder tractar-les, aquestes depenien

d'una sèrie de funcions, les quals s'emmagatzemen al mòdul "LlistaPDI". Molts dels problemes que vam tindre amb les llistes, van ser deguts a que, a l'hora d'implementar les diferents funcions específiques de les llistes, no vam tindre en compte el funcionament de cadascuna d'elles per separat. Un exemple d'aquest cas es va donar a l'hora d'implementar la funció "FerCursa", mitjançant la qual anàvem modificant els punts i kilòmetres de tots el pilots de la llista "lp". En aquest cas fèiem servir un bucle general que, extreia la informació del pilot seleccionat pel "PDI", eliminàvem la seva informació de la llista, i el tornàvem a inserir un cop modificat. En aquest cas, no vam tindre en compte, el lloc on s'ubicava el "PDI" al executar la comanda "LLISTAPDI_P_esborra", i com que més endavant avançàvem el "PDI", ens saltàvem un pilot a cada lectura, provocant que només es tinguessin en compte els pilots en posicions imparelles, és a dir: el primer, el tercer, el cinquè,...

Per altre banda, el problema que ens vam trobar amb la utilització de la comanda "rand()", va ser que la fèiem servir en funció del temps, encara que en cap moment marcàvem un temps inicial, amb el qual, el valor aleatori obtingut sempre era 0. És per això que, en comptes de deixar-lo en funció del temps, vam fer que no depengués de res, forçant a que ens proporcionés un número enter aleatori, que després dividíem entre 10.

Aquesta pràctica sobretot ens ha demostrat que cal tindre en compte fins al més mínim detall a l'hora de programar, ja que, com ens ha passat, tots els elements tenen influència en la resta, alterant així el resultats que s'esperen del programa. Tal i com vam veure ja a la fase1, escriure codi sense pensar primer en el funcionament bàsic de la pràctica, porta molts problemes que suposen una gran pèrdua de temps que no ens podem permetre. El fet de fer-se esquemes de la pràctica, o be apuntar-se en un full apart el funcionament del procediments i funcions, sobretot de cara a la manipulació d'estructures, és de gran ajuda quan s'han de realitzar treballs extens com ara aquesta pràctica. Aquests petits detalls optimitzen el treball i ajuden, sobretot, a prevenir futurs problemes com ara el esmentat anteriorment.