Proyecto 2ºSemestre

Alumnos		Grupo
Nombres	**Apellidos**	
Wesley	Lucas Mas	**Nota**

Fecha Entrevista	Código	Firma Estudiante	Firma Becario

Copia alumnos

Alumnos		Grupo
Nombres	**Apellidos**	
Wesley	Lucas Mas	**Nota**

Fecha Entrevista	Código	Firma Estudiante	Firma Becario

Comentarios

Copia becarios

Índice

1. Resumen del enunciado + Explicación del diseño del proyecto

El proyecto consiste en el desarrollo de un juego de temática de carreras de Fórmula 1, donde no solo tenemos que aplicar conceptos ya dados previamente en el semestre 1, sino que además se han de dar conceptos aprendidos a lo largo del semestre 2 como es el tratamiento de ficheros, uso de punteros, petición, tratamiento y liberación de memoria dinámica, comprensión y uso de estructuras de datos lineales, y por último, saber hacer uso de una IDE como CLion para saber no solo a trabajar en un ámbito distinto del terminal de LaSalle, sino además para hacer uso de librerías ajenas que no hemos usado previamente como es el caso del LS_allegro, que nos permite en esta ocasión poder ejecutar ventanas gráficas donde poder mostrar toda la información que queramos de forma gráfica como si se tratase en este caso, de un videojuego.

En cuanto al diseño de este proyecto, la idea ha sido crear un total de 7 módulos, junto con el de LS_allegro que se nos ha proporcionado, donde estructuramos cada funcionalidad importante del programa en distintas partes encapsuladas. La forma en la que se ha trabajado en todo momento ha sido primero escribir el código dentro del main para evitar posibles problemas con el tratamiento de módulos al comenzar a entenderlos, y una vez entendido dicho tema, pasar todo el código elaborado a un módulo donde sobretodo se ha puesto a prueba que funcione de forma debida, en este caso por temas como pasar punteros por referencia/valor, etc…
Sobre todo, la idea general del diseño de este proyecto va enfocada en dividir las partes más importantes en diferentes módulos. De esta forma conseguimos obtener un código limpio y ordenado donde incluso a ojos de una persona que no es un programador/a entienda almenos cómo está estructurado todo.

2. Estructura de la práctica

2.1. Explicación de los módulos

- **“estructuras.h”**

  Creado única y exclusivamente como lugar de reunión de todos los tipos definidos que nos ayudan a almacenar información a lo largo del programa.

- **“ficheros.h”**

  Creado única y exclusivamente solo para el tratamiento de los ficheros que introducimos como argumentos. Es decir, aparte de abrirlos y ver si están vacíos, también nos sirve para extraer toda la información que contienen dentro y luego almacenarla en diferentes tipos creados exclusivamente en el fichero de "estructuras.h".

- **“LS_allegro.h”**

  Modulo proporcionado por los becarios para arrancar las funciones elementales que nos permiten jugar con las ventanas graficas a lo largo del programa.
  Es un módulo diseñado por Albert Lloveras Carbonell, el cual es de simplificación y adaptación del API de la librería Allegro 5 para el desarrollo sencillo de juegos 2D en la asignatura de Programación I. Esta adaptación solo facilita el desarrollo ahorrando al alumno de tener conocimientos previos avanzados sobre listeners i events que utiliza esta librería para realizar tareas como el escucha del teclado, la iniciación o el muestreo por pantalla.

- **“menu.h”**

  Creado única y exclusivamente para encapsular el menu del juego. Es decir, con tal de reducir el número de líneas del main.c (o PJ2_wesley.lucas.c), he creado este módulo para envolver toda la funcionalidad del menu, ya sea mostrar las opciones, meter la funcionalidad de cada una de estas y además de generar el fichero log.txt de la opción 4. Remarcar que no he implementado un módulo de la opción 4, ya que, a diferencia de los otros 3 módulos, esta contiene muy pocas líneas de codigo (para ser exactos, 17 líneas) y no tendría sentido crear un módulo exclusivamente para esta.
  P.D. No sé si está bien visto este aspecto, pero he introducido bastantes argumentos de entrada en el procedimiento de menuExe. He intentado cambiar la lógica, reducir el número de variables, etc... Pero por más que pensase en cómo llevarlo a cabo, no solo perdía más tiempo al reformularlo de distintas maneras, sino que atrasaba el desarrollo del programa. Principalmente, mi intención con este proyecto ha sido hacerlo lo óptimo posible a nivel de líneas de codigo.

- **“opcion1.h”**

  Creado única y exclusivamente para la opción 1 del programa. Desde almacenar la información de petición de información del usuario en el registro 'OpcionUno', hasta mostrar el configurador del programa donde podemos escoger los diferentes tipos de pieza, motores que nos sea de interés para competir más adelante en la simulación de la carrera en la opción 2.

- **“opcion2.h”**

  Creado única y exclusivamente para la opción 2 del programa. Aquí se empieza a trabajar densamente con la librería de LS_allegro. Esta sobretodo enfocado en jugar y saber manejar los valores de la información extraída de los ficheros con tal de simular una carrera de coches. Y para ello requiere además saber hacer buen manejo y control de las variables/tipos con punteros.

- **“opcion3.h”**

  Creado única y exclusivamente para la opción 3 del programa. Aquí principalmente se trabaja en ordenar las posiciones de los pilotos en base a dos factores: el tiempo y los puntos acumulados en cada GP. Se nos pedirá ejecutar una ventana gráfica con los resultados de ranking en base a estos dos factores previamente mencionados, y para llevarlo a cabo, se tendrá que trabajar y manejar con la memoria dinámica de todas las variables con las que se han trabajado.

- **“sortedlist.h”**

  Creado única y exclusivamente con la intención de ordenar los grandes premios en base a su turno correspondiente de ejecución cada vez que se comience una nueva carrera. Es decir, esta estructura de datos nos ayuda a ir pasando de grande premio en grande premio cada vez que empecemos o acabemos alguna de las carreras a realizar.

2.2. Explicación de cada función dentro del módulo correspondiente

- **“estructuras.h”**
  - Ninguna función. Solo definiciones de tipos propios usados a lo largo de todo el programa.

- **“ficheros.h”**
  - FILE *aperturaFicheroTxt (char *arg_fichero);

    Abrir un fichero de texto. Sobretodo comprueba que no haya un error de procesamiento de fichero.

  - FILE *aperturaFicheroBin (char *arg_fichero);

    Abrir un fichero binario. Sobretodo comprueba que no haya un error de procesamiento de fichero.

  - int compruebaEstadoFichero (FILE *punt_fich, int tamano);

    Comprobar si el fichero sobre el que se lee contiene información o no.

  - InfoPieza *guardadoInfoPiezas (FILE *p, int *num_piezas);

    Leer el fichero de texto de piezas y almacenar toda la información que se encuentra dentro de este en un tipo creado específicamente para este, llamado 'InfoPieza'.

- InfoGPs *guardadoInfoGPs (FILE *gps, int *num_premios);

  Leer el fichero de texto de piezas y almacenar toda la información que se encuentra dentro de este en un tipo creado específicamente para este, llamado 'InfoGPs'.

- InfoCorredor *guardadoInfoCorredores (FILE *c, int num_pilotos);

  Leer el fichero de texto de corredores y almacenar toda la información que se encuentra dentro de este en un tipo creado específicamente para este, llamado 'InfoCorredor'.

- InfoBase *guardadoInfoBase (FILE *b, int num_coches);

  Leer el fichero de texto de piezas y almacenar toda la información que se encuentra dentro de este en un tipo creado específicamente para este, llamado 'Pieza'.

- void cierreFichero (FILE *p);

  Cerrar el fichero sobre el que ya hemos operado previamente.

- **"LS_allegro.h"**
  - int LS_allegro_init(int nAmplitud,int nAltura,char *sNomFinestra);

    Devuelve 1 (CIERTO) si se ha podido inicializar correctamente el Framework de Allegro5. En caso contrario se devolverá 0 (FALSO).

  - int LS_allegro_key_pressed(int nKey);

    Devuelve 1 (Cierto) si se ha pulsado la tecla recibida al parámetro nKey. En caso contrario, se devolverá 0 (FALSO). ¡¡ATENCIÓN!! ¡LECTURA DESTRUCTIVA!

  - ALLEGRO_COLOR LS_allegro_get_color(int nColor);

    Devuelve una variable del tipo ALLEGRO_COLOR correspondiente al color solicitado.

  - ALLEGRO_FONT* LS_allegro_get_font(int nSize);

    Devuelve una variable del tipo ALLEGRO_FONT * para poder usar la función al_draw_textf más fácilmente.

  - void LS_allegro_exit();

    Libera la memoria que se había reservado para las variables

necesarias para hacer funcionar Allegro5.

- void LS_allegro_clear_and_paint(int nColor);

  Fija el color de fondo de la ventana gráfica.

- void LS_allegro_console_fflush();

  Limpia los buffers del teclado para que no quede ninguna tecla almacenado.

- void LS_allegro_console_clear_screen();

  Limpia la pantalla de la consola

- **"menu.h"**
  - void menuExe (OpcionUno opc_1, OpcionDos *opc_2, OpcionTres *opc_3, InfoPieza *pieza, InfoCorredor *corredor, InfoBase *stats_def, SortedList list, GPese node_opc2, StructSuma *suma, DatosTotal *suma_p, int num_piezas, int num_premios, int num_pilotos);

    Agrupar y ejecutar todas las funciones que van desde la opcion 1 hasta la 4 del programa.

- **"opcion1.h"**
  - OpcionUno getOpc1Info (OpcionUno opc_1, InfoPieza *pieza, int num_piezas, int *opc1_in);

    Llenar las variables del tipo propio 'OpcionUno', además de reproducir la ventana grafica de selección de piezas y motores de preferencia.

- **"opcion2.h"**
  - OpcionDos * getOpc2Info (OpcionDos *opc_2, OpcionUno opc_1, int num_pilotos, int num_piezas, InfoPieza *pieza, InfoBase *stats_def, StructSuma *suma, GPese node_opc2, InfoCorredor *corredor, int j);

    Llenar las variables del tipo propio 'OpcionDos', además de reproducir la ventana grafica de simulación de la carrera los resultados de nuestro piloto respecto a la posición del ranking en la que se posiciona.

- **"opcion3.h"**
  - OpcionTres *displayOpc3Results (OpcionTres *opc_3, OpcionDos *opc_2, int j, int num_pilotos, int opc2_in);

Mostrar el ranking de mayor a menor tiempo de completar la carrera de cada GP participado.

- DatosTotal *sortPoints (DatosTotal *suma_p, OpcionTres *opc_3, OpcionUno opc_1, InfoCorredor *corredor, int *j, int num_pilotos, int num_premios);

  Llenar el tipo propio 'DatosTotal' de la información esencial a mostrar luego en los resultados finales del ranking de la temporada finalizada.

- void displayPointsRanking (DatosTotal *suma_p, int j, int num_pilotos);

  Mostrar el ranking de la clasificación final de temporada en base al número de puntos acumulado en cada grande premio participado de cada corredor.

- **"sortedlist.h"**
  - SortedList SORTEDLIST_create ();

    Crea una lista ordenada vacía. Si la lista no puede crear el nodo fantasma, se establecerá el código de error a LIST_ERROR_MALLOC.

  - void SORTEDLIST_sortedAdd (SortedList * list, InfoGPs gps_, int num_premios);

    Inserta el elemento especificado en esta lista en la posición definida por el algoritmo de clasificación. Cambia el punto de vista del elemento (si lo hay) y cualquier elemento posterior a la derecha. Si la lista no puede crear el nuevo nodo para almacenar el elemento, establecerá el código de error en LIST_ERROR_MALLOC.

  - void SORTEDLIST_remove (SortedList * list);

    Elimina el elemento actual en el punto de vista de la lista. Desplaza cualquier elemento posterior a la izquierda. Esta operación fallará si el PDV es posterior al último elemento válido de la lista. Eso también sucederá para una lista vacía. En esa situación, esta operación establecerá el código de error en LIST_ERROR_END.

  - GPese SORTEDLIST_get (SortedList * list);

    Devuelve el elemento actual en el punto de vista de la lista. Esta operación fallará si el PDV es posterior al último elemento válido de la lista. Eso también sucederá para una lista vacía. En esa situación, esta operación establecerá el código de error en LIST_ERROR_END.

- int SORTEDLIST_isEmpty (SortedList list);

  Devuelve 'verdadero' (! 0) si esta lista no contiene elementos.

- void SORTEDLIST_goToHead (SortedList * list);

  Mueve el punto de vista al primer elemento de la lista.

- void SORTEDLIST_next (SortedList * list);

  Mueve el punto de vista al siguiente elemento de la lista. Si el PDV es posterior al último elemento de la lista (o cuando la lista está vacía), esta función establecerá el error de la lista a LIST_ERROR_END.

- int SORTEDLIST_isAtEnd (SortedList list);

  Devuelve 'verdadero' (! 0) si el POV es posterior al último elemento en la lista.

- void SORTEDLIST_destroy (SortedList * list);

  Elimina todos los elementos de la lista y libera cualquier bloque de memoria dinámica que la lista estaba usando. La lista debe crearse nuevamente antes de su uso.

- int SORTEDLIST_getErrorCode (SortedList list);

  Esta función devuelve el código de error proporcionado por la última operación ejecutada. Las operaciones que actualizan el código de error son: Crear, agregar, eliminar y obtener.

3. Estructuras de datos

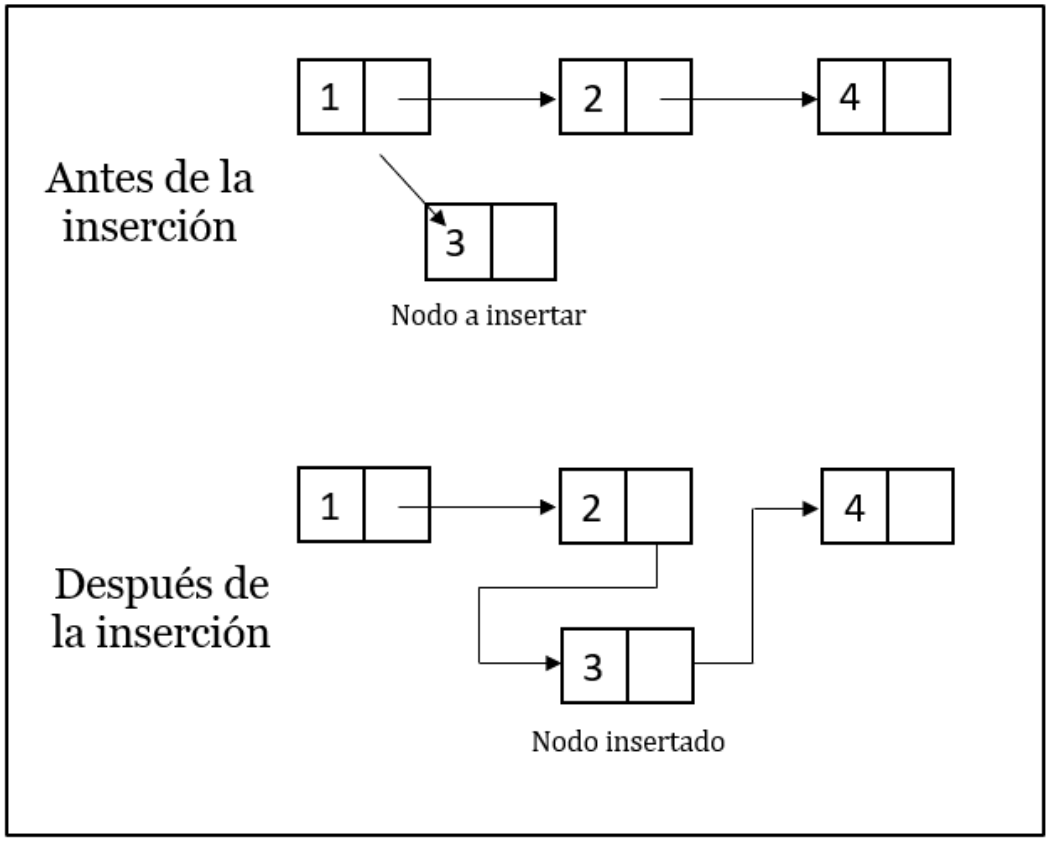
3.1. Explicación de las estructuras de datos utilizadas

La estructura de datos que se nos ha encomendado utilizar en esta ocasión para el proyecto de 2º semestre ha sido nada más y nada menos que una lista ordenada. Esta se caracteriza principalmente por crear tantos nodos como el usuario desee con tal de que cada nodo guarde un tipo de información que sea clave para luego ordenar según el valor que contiene dentro de este. Es decir, si por cada nodo hay un número del 1 al 10 y estos están desordenados (que entonces eso sería una linked list al estar desordenados), el trabajo principal de la lista ordenada es ordenar esos mismos elementos, es decir, los números del 1 al 10, de menor/mayor a mayor/menor rango hasta que queden ordenados tal y como el usuario desea.

3.2. Justificación de la elección

Se utiliza una lista ordenada principalmente para la implementación de una parte de la opción 2 del programa. Es decir, en la opción 2 se nos pide ir gran premio por gran premio para realizar cada simulación de carrera correspondiente además de realizar una serie de cálculos que luego hemos de guardar en tipos propios. Con tal de ir avanzando gran premio por gran premio, como bien iba diciendo, se puede recurrir a un tipo propio que avanzase a través de una variable índice, por ejemplo. Pero en esta ocasión, al ver que las estructuras de datos lineales ofrecen una serie de funciones predeterminadas a utilizar que nos permiten avanzar, retroceder, eliminar, destruir, etc… nos es pues más sencillo utilizar una lista ordenada para guardar la posición de cada grande premio correspondiente. Es decir, cada grande premio tiene un turno de comienzo. El orden que se nos proporcionaba era 1-2-4-3, y el objetivo era ordenarlo dejándolo en 1-2-3-4 para seguir un orden.

3.3. Diagrama de actividades del funcionamiento de la estructura utilizada



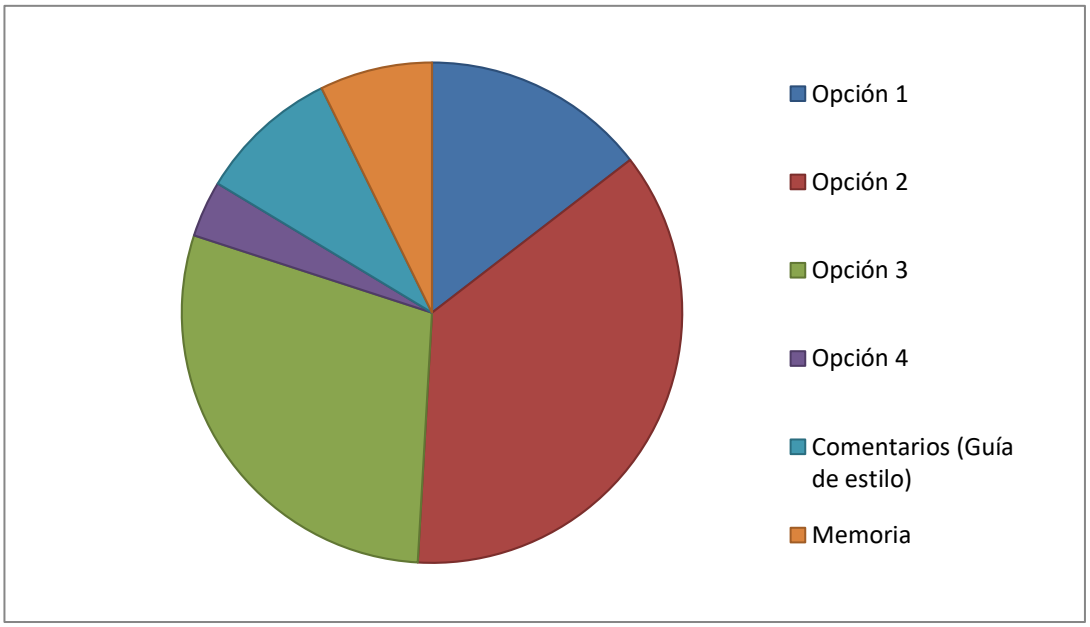
4. Problemas observados y cómo se han solucionado

He tenido varios problemas de diversas magnitudes a lo largo del desarrollo de este programa.
Para comenzar, están los problemas de la librería de LS_allegro, que mayoritariamente eran causados debido a que solapaba un elemento con otro en la misma área de ubicación o similar.
Luego también he tenido problemas de acceso a la memoria, que se debían a dos razones:

1. En el malloc al pedir la memoria para algún tipo propio donde había información para los 7 corredores y nuestro piloto, en vez de indicar 'num_pilotos + 1', indicaba únicamente 'num_pilotos', y eso a la larga creaba problemas de acceso a la memoria a la hora de printar alguna información de una casilla en concreto o incluso al operar con dichas variables.
2. Al crear subtipos propios dentro de tipos propios principales, tenía obviamente que pedir memoria dinámica para estos. Y para hacerlo, tenía que incorporar un bucle para cada casilla del array algo de espacio de memoria. Esto provocaba que al acabar de pedir memoria dinámica para cada sub-tipo propio, no cerraba el bucle y dejaba el programa continuase funcionando sin haber acabado de cerrarlo. Ha habido muchos momentos donde actuaba con normalidad a pesar de que estuviese mal. Aun así, Joan Fitó comentó que era una práctica incorrecta, ergo, que cerrase el bucle cada vez que pedía memoria dinámica, y ya está.

También he tenido errores relacionados con cerrar de forma errónea los ficheros y liberar la memoria. Esto se solucionaba mirando donde acababa cada indentación, y antes de que se cerrase la indentación de apertura de un fichero o petición de memoria, tenía que indicar el free o flcose correspondiente.
Y, por último, uno de los problemas que me llevó más tiempo de solucionar era la comprensión de la lógica detrás de la estructura de datos lineal de la lista ordenada. A la hora de entenderlo me basé en los códigos de la lista ordenada que nos proporcionaban los profesores, y donde más me fallaba la comprensión era en la función de sortedAdd. Digamos que antes de entrar a esta función ya ordenaba toda la información previamente y luego le pedía dentro de sortedAdd que me hiciese un linkedAdd para meter la información dentro del nodo, cuando en realidad se tenía que ordenar dentro de la función sortedAdd sin recurrir a la lógica que yo tenía en mente en ese momento.

5. Tiempo de realización de la práctica



Para comenzar, con la opción 1 el único problema que ha habido ha sido la parte lógica de guardar la pieza seleccionada con sus respectivos motores también seleccionados. Al principio pensaba que el fichero de piezas que se nos proporcionaba no iba a ser modificado luego a puertas cerradas a la hora de corregir la práctica, pero no era así, y fue una de las partes del programa donde a nivel lógico me costaba entender cómo guardar las piezas según las variables de índice con las que trabajaba.

Por otra parte, también se encuentra la opción 2, con diferencia, la que más he tardado en realizar debido a la lógica del control del tiempo a la hora de mover los coches, encender/apagar las luces de los semáforos y sobretodo ordenar la información en diferentes tipos con tal de que a la larga, de lo que quedaba de desarrollo de proyecto, me fuese más severo y fácil lidiar con los valores guardados.

Después la opción 3, que se gana el segundo puesto en el pódium de dificultad. Fue a partir de aquí donde empecé a tener problemas de acceso a la memoria y donde tuve que ir cambiando más entre la opción 2 y esta con tal de ver los resultados que me daban los printfs de debug que me creaba a lo largo del programa con tal de ver qué iba mal y la causa de por qué. Por no decir que también he tenido mis problemas con el LS_allegro como ya comenté previamente en el apartado de 'Problemas observados y cómo se han solucionado'.

Luego está la opción 4. Su implementación no tardó más de media hora en llevarse a cabo, ya que toda la información a guardar en el log ya la tenía guardad en tipos propios, así que hacer el paso de información a fichero de texto no era nada difícil.

Y, por último, está la guía de estilo y la memoria. Comenzando por la guía de estilo, al estar trabajando con módulos y creando funciones constantemente, era obligatorio comentar todo lo desarrollado con tal de mantener un nivel de formalidad en el fichero main (PJ2_wesley.lucas.c) de nuestro programa. Lo cual, quiere decir que

requería comentar funciones, módulos, defines, tipos propios, líneas de código de comprensión compleja a nivel de código, etc… Definitivamente, al ser un proyecto donde se incluye más conceptos a introducir, obviamente implica añadir más comentario que se explique debidamente del por qué se hace así. Y por último la memoria, se ha redactado en una tarde y no tenía mucha dificultad de desarrollo, ya que en principio se está relatando la experiencia del alumno al afrontar un proyecto de este calibre de la asignatura de Programación I.

6. Contenido extra

- Se ha añadido en la pantalla de carga de los semáforos a los cuatro becarios de prácticas de la asignatura de programación moviéndose de una punta de la pantalla hasta la otra. De esta forma no solo hacía amena la espera de carga de la simulación de la carrera, sino que ponía en práctica el control del tiempo moviendo objetos antes de comenzar con la parte lógica del movimiento de los coches.

- Dudo que esto cuente mucho como 'contenido extra' pero por si acaso lo comento. En la pantalla de vista/simulación de la carrera, he añadido al lado del contador de pit stops una imagen en formato .png de un personaje cómico de YouTube llamado Joji (AKA Filthy Frank). En relación a los tiempos de espera de los pit stops…

- En la opción 3 del programa, al acabar todos los grandes premios, si se pulsa la tecla D, estando actualmente en el 4º grande premio, el programa debería llevarnos a un ranking que califica a los corredores de menor a mayor rango según los puntos obtenidos acumulados en cada grande premio. En mi caso, en vez de hacerlo así, lo he hecho de tal forma que, habiendo acabado el 4º premio, seleccionamos la opción 3, nos dirige al ranking según el tiempo que han tardado los corredores, presionamos la tecla 'ESC' y nos lleva a otra pantalla donde nos muestra el ranking según el número de puntos acumulados. Al presionar 'ESC' otra vez y seleccionar la opción 3, nos lleva al ranking según los puntos obtenidos.
  Lo he hecho de esta forma porque según tengo entendido algunos de los becarios del equipo de prácticas de Programación dijeron que la gente lo hacía de distintas formas, pero que era aceptable siempre y cuando funcionase correctamente la muestra de resultados. Incluso he mirado en el documento de la Rubrica y no queda contemplado la posibilidad de bajar nota ante una alternativa propuesta como la mía. Incluso he hablado con el becario Joan Fitó y en principio si no queda contemplado, lo peor que puede pasar es que se bajen 0,25 puntos de dicho apartado, dependiendo según del criterio de cada becario.

- Los comentarios de los módulos, funciones, líneas de código, etc… se han escrito sin tener en cuenta las reglas otográficas, ya que, al pasar un fichero de una IDE a otra, temo que provoque una lectura difícil para el becario o usuario que la esté leyendo.

- Los comentarios de los módulos de LS_allegro.h y .c han permanecido intactos en todo momento por su autor original, y luego en el módulo de sortedlist.h y .c han permanecido intactos también los comentarios en inglés originales escritos por los profesores de Programación I, además de añadir yo por mi cuenta mi propia cabecera de propósito, autor y fechas de creacion y última modificación.

- En la guía de estilo se especifica como consejo que el fichero sobre el que se trabajó en todo momento reciba el nombre de 'main.c'. En mi caso, como yo no sabía si este consejo se seguía de raja a tabla, y tampoco porque en el apartado de fechas de entrega no se especifica la extensión .c, he dejado como nombre de mi main.c el nombre del proyecto que es 'PJ2_wesley.lucas.c'. Lo comento por si acaso no haya ningún malentendido.

- Más que 'contenido extra' lo pongo como una observación lo que comentaré ahora. A lo largo del desarrollo del programa he intentado abreviar el número de líneas del main lo máximo que he podido sin que se me produjese inconvenientes de por medio. En lo que respecta a las funciones de módulos y funciones normales, sin contar las líneas de comentarios y siguiendo la guía

de estilo, me ocupan las funciones hasta un máximod de 75 líneas. Y sí, ya sé que queda contemplado en la guía de estilo que sean un total de 60, pero al haber estado hablando con Joan Fitó sobre este tema, comentó que no era más que una **sugerencia**. Aun así, de no seguir dicha sugerencia, no sería un suspenso automático, pero sí se bajaría la nota a nivel de décimas.
Es decir, con esto lo que quiero decir es que he optimizado el código lo máximo posible a nivel de líneas entre las 60 y 75, excepto en el main, que se complicaba las cosas al tener que pasar muchas variables por argumentos de cabecera de función y a la larga era un caos por tema de punteros y el programa me falla en accesos de memoria. El main, quitando comentarios y siguiendo la guía de estilo, se me ha quedado en un total de 150 líneas aprox. En esas 150 líneas se encuentran:

- Apertura de ficheros y petición de memoria dinámica
- La función de menuExe que engloba un tercio del funcionamiento de toda la práctica.
- Y, por último, la liberación de memoria y el cierre de ficheros.

No dudo en que se podía haber metido en una función general el apartado de 'Apertura de ficheros y petición de memoria dinámica' y en otra función la 'liberación de memoria y el cierre de ficheros'. De esta forma se me podía haber quedado el código resumido en unas meras 15 líneas. Pero lo he intentado por activa y por pasiva poder solucionar este problema de optimización y no ha podido ser por dos razones:

1. Por los errores que he comentado previamente que me causaba el programa relacionados con el acceso a la memoria en varias variables-
2. En la situación actual en la que nos encontramos con el coronavirus, ha provocado que almenos en el departamento de Ingeniería Multimedia nos hayan metido exceso de trabajo con tal de justificar la falta de punto de control en este semestre. Provocando pues que se nos evaluasen por muchas prácticas en varias asignaturas mientras que a la vez había otras donde no solo habían prácticas, sino también habían exámenes.
   Con esto no intento justificarme y escudarme detrás de la organización que lleva el propio departamento de mi grado del por qué no he podido optimizar el número de líneas en el main.c del programa. Sin embargo, sabiendo que voy solo en este proyecto y que no ha habido ni un solo día en el que no me pasase por el aula virtual de dudas de programación, espero que almenos se pueda entender que a pesar de este problema de optimización, espero que no signifique una reducción de puntos bastante férrea o incluso un suspenso, sobre todo porque Programación I se basa más en hacer funcionar el programa, en vez del enfoque de la optimización de este, ya que de eso se encarga

la asignatura de segundo año de DPOO por lo que me he informado y me han comentado compañeros y profesores.

7. Conclusiones

Este proyecto me ha ayudado a entender de principio a fin todos y cada uno de los conceptos aprendidos a lo largo del semestre. Es decir, comenzando primero por el tratamiento de ficheros a nivel de lectura y extracción de datos hasta la opción 4 donde teníamos que escribir sobre este, me ha ayudado a no solo poner en práctica los conceptos de ficheros, sino que además me ha ayudado a resolver dudas referentes a este y la lógica detrás de cada funcionamiento de cada función relacionada con dicho concepto.

Lo mismo va para la comprensión de la lógica de la lista ordenada. Fue uno de los conceptos que más me tiempo me llevó a entender bien a la hora de empezar a desarrollar la opción 2 del programa, y no solo me ha servido paraa entender esta estructura de datos lineales, sino también a comprender mejor otras como la FIFO, LIFO (estáticas y dinámicas), LinkedList, etc… Ya que comparten conceptos entre ellas que facilitan la comprensión de su debido funcionamiento y lógica correspondiente.

Por otra parte, este proyecto me ha servido para trabajar y adaptarme en un entorno de trabajo distinto al servidor de matagalls de LaSalle, que en este caso es la IDE de CLion. Al principio tenía mis problemas de adaptación, sobre todo porque funciones como el 'atoi' provocaba que tuviese que recurrir a un atoi casero en según qué circunstancias. Pero en general, ha sido un mejor entorno de trabajo en comparación con el que teníamos el año pasado con CodeLite realizando incluso la partición de disco incluso para aquellos que usábamos Windows…

Por último, y en relación con el punto previo, me ha servido este proyecto para entender y saber trabajar con librerías ajenas como LS_allegro, sobretodo me ha servido para entender cómo funcionaban los módulos y qué relación había entre las funciones compartidas entre el main, el .c y el .h de dicho módulo.

En general, se podría decir que he extraído el conocimiento suficiente como para saber cómo estructurar un código de forma correcta con los módulos, saber trabajar con otras librerías ajenas. A ver buen manejo de estructuras de datos lineales, etc…

Y eso no solo se ha conseguido gracias a la persistencia mantenida a lo largo del semestre con el desarrollo de este programa, sino también gracias al equipo de becarios e intensificadores de Programación I que han sabido trasladar de forma eficiente las sesiones de dudas de la asignatura al formato online, lo cual se agradece que hayan mantenido su compromiso a lo largo del curso sabiendo que ha habido problemas de becas con la institución de LaSalle en este segundo y último semestre debido a la crisis del coronavirus.