Práctica 2: Lista de tareas (parte 2)

Programación 2

Curso 2020-2021

En esta práctica se ampliarán las funcionalidades de la *Práctica 1*, añadiendo la gestión de múltiples proyectos, opciones para importar y exportar ficheros de texto, guardar y recuperar datos de un fichero binario, y la posibilidad de pasar argumentos al programa por línea de comando. Los conceptos necesarios para desarrollar esta práctica se trabajan en el *Tema 1*, *Tema 2* y *Tema 3* de teoría.

Condiciones de entrega

- La fecha límite de entrega para esta práctica es el **viernes 16 de abril**, hasta las **23:59**
- Debes entregar un único fichero llamado `prac2.cc` con el código de todas las funciones

Código de honor

Si se detecta copia (total o parcial) en tu práctica, tendrás un **0** en la entrega y se informará a la dirección de la Escuela Politécnica Superior para que adopte medidas disciplinarias

Está bien discutir con tus compañeros posibles soluciones a las prácticas
Está bien apuntarte a una academia si sirve para obligarte a estudiar y hacer las prácticas

Está mal copiar código de otros compañeros para resolver tus problemas
Está mal apuntarte a una academia para que te hagan las prácticas

Si necesitas ayuda acude a tu profesor/a
No copies

Normas generales

- Debes entregar la práctica exclusivamente a través del servidor de prácticas del Departamento de Lenguajes y Sistemas Informáticos (DLSI). Se puede acceder a él de dos maneras:
  - Página principal del DLSI (`https://www.dlsi.ua.es`), opción “ENTREGA DE PRÁCTICAS”
  - Directamente en la dirección `https://pracdlsi.dlsi.ua.es`
- Cuestiones que debes tener en cuenta al hacer la entrega:
  - El usuario y la contraseña para entregar prácticas son los mismos que utilizas en UACloud
  - Puedes entregar la práctica varias veces, pero sólo se corregirá la última entrega
  - No se admitirán entregas por otros medios, como el correo electrónico o UACloud
  - No se admitirán entregas fuera de plazo

- Tu práctica debe poder ser compilada sin errores con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas
- Si tu práctica no se puede compilar su calificación será 0
- El 70 % de la nota de la práctica dependerá de la corrección automática, por lo que es imprescindible que respetes estrictamente los textos y los formatos de salida que se indican en este enunciado. El otro 30 % dependerá de la revisión manual del código que haga tu profesor de prácticas, por lo que debes también ajustarte a la guía de estilo de la asignatura. Además se tendrá en cuenta que hayas actualizado tu código en *GitHub* todas las semanas
- Al comienzo de todos los ficheros fuente entregados debes incluir un comentario con tu NIF (o equivalente) y tu nombre. Por ejemplo:

prac2.cc

```
// DNI 12345678X GARCIA GARCIA, JUAN MANUEL
...
```

- El cálculo de la nota de la práctica y su relevancia en la nota final de la asignatura se detallan en las transparencias de presentación de la asignatura (*Tema 0*)

1. Descripción de la práctica

Mientras que en la *Práctica 1* se gestionaba un único proyecto, en esta práctica se gestionará una lista de proyectos. Se añadirán elementos al menú de la práctica anterior para añadir y borrar proyectos, además de nuevas opciones para la importación/exportación de datos desde ficheros de texto y la lectura/escritura de ficheros binarios.

2. Detalles de implementación

En el Moodle de la asignatura se publicarán varios ficheros que necesitarás para la correcta realización de la práctica:

- `newErrorFunction.cc`. Este fichero contiene una versión actualizada del tipo enumerado `Error`, con nuevos errores que se pueden dar en esta práctica, como por ejemplo `ERR_FILE`. También contiene una actualización de la función `error`, con los mensajes correspondientes a los nuevos tipos de error introducidos. Copia el contenido de este fichero en tu código de la *Práctica 1* sustituyendo al tipo enumerado `Error` y a la función `error` que tenías antes
- `autocorrector-prac2.tgz`. Contiene los ficheros del autocorrector para probar la práctica con algunas pruebas de entrada. La corrección automática de la práctica se realizará con un programa similar, con estas pruebas y otras más definidas por el profesorado de la asignatura
- `prac2`. Fichero ejecutable de la práctica (compilado para máquinas Linux de 64 bits) desarrollado por el profesorado de la asignatura, para que puedas probarlo con las entradas que quieras y ver la salida correcta esperada

- A diferencia de la *Práctica 1*, para cada una de las nuevas opciones que se van a añadir al menú del programa deberás decidir tú los nombres de las funciones que vas a utilizar, sus parámetros y los valores que devuelven
- En la corrección de la práctica se introducirán siempre datos del tipo correcto, aunque con valores que pueden ser incorrectos. Por ejemplo, si se pide el tiempo de duración de una tarea, se probará siempre con un valor entero, que podría ser `-1237` o `0`, pero nunca se introducirá un valor de otro tipo (carácter, `string`, número real, ...)

- El resto de decisiones en la implementación de la práctica quedan a tu criterio, pero ten en cuenta que el código fuente será revisado por tu profesor de prácticas siguiendo la guía de estilo publicada en el Moodle de la asignatura. Parte de la nota de la práctica depende de dicha revisión

3. Componentes

Además de los registros que se definieron en la *Práctica 1* (`Date`, `Task`, `List` y `Project`), se deben definir cinco nuevos tipos de registros para poder almacenar información de múltiples proyectos y trabajar con ficheros binarios. Esto último es necesario porque hay registros de la *Práctica 1* que contienen campos de tipo `string` y/o `vector` que, como sabes, no pueden utilizarse para leer o escribir información de ficheros binarios al no tener un tamaño fijo (si no sabes de que estoy hablando, consulta la transparencia 33 del *Tema 3* de teoría).

Los registros `BinTask`, `BinList`, `BinProject` y `BinToDo`, que definiremos a continuación, sólo se utilizarán cuando haya que leer o escribir en un fichero binario. Para el resto de funcionalidades de la práctica se seguirán usando los registros `Task`, `List`, `Project`, como se hacía en la *Práctica 1*, junto con el nuevo registro `ToDo` para gestionar múltiples proyectos. Para el registro de tipo `Date` no es necesario definir un tipo nuevo para trabajar con ficheros binarios, ya que al contener únicamente valores enteros se puede utilizar sin problema con este tipo de ficheros. Los siguientes apartados describen la estructura de cada uno de estos registros en detalle.

3.1. ToDo

Para poder gestionar múltiples proyectos en esta práctica, es necesario definir un nuevo tipo de registro:

```
struct ToDo{
  int nextId;
  string name;
  vector<Project> projects;
};
```

Este registro tiene un nombre (`name`) y un vector (`projects`) que almacenará todos los proyectos que se vayan creando en la aplicación. También contiene un campo entero `nextId` que almacenará el identificador que deberá asignarse al campo `id` del siguiente proyecto que se cree. Este campo tendrá el valor `1` al inicio del programa y se incrementará de uno en uno para cada nuevo proyecto. Es decir, el primer proyecto que se cree tendrá como identificador el `1`, el segundo tendrá el `2` y así sucesivamente.

En el programa principal (`main`) deberás declarar una variable de tipo `ToDo` y asignar a su campo `name` el valor "`My ToDo list`" y a su campo `nextId` el valor `1`. El vector `projects` estará inicialmente vacío e irá creciendo conforme se añadan nuevos proyectos al programa.

- Ten en cuenta que la variable de tipo `ToDo` que vas a crear deberá almacenar todos los proyectos que se creen en el programa. Vas a tener que modificar la función `main` no solo para añadir las nuevas opciones del menú, sino también para poder gestionar adecuadamente el trabajo con esta variable

3.2. BinTask

Este tipo de registro se utilizará para leer y escribir información de una tarea en el fichero binario:

```
struct BinTask{
  char name[KMAXNAME];
  Date deadline;
  bool isDone;
  int time;
};
```

Solamente se ha introducido un cambio con respecto al registro `Task`: el campo `string name` se sustituye por el campo `char name[KMAXNAME]`, donde `KMAXNAME` es una constante de tipo entero cuyo valor será 20.

3.3. BinList

Este tipo de registro se utilizará para leer y escribir información de una lista de tareas en el fichero binario:

```
struct BinList{
  char name[KMAXNAME];
  unsigned numTasks;
};
```

Los cambios que se han introducido con respecto a `List` son los siguientes:

- Como en `BinTask`, el campo `string name` se sustituye por el campo `char name[KMAXNAME]`
- Aparece un campo `unsigned numTasks` que indica el número de tareas que contiene la lista
- El campo `vector<Task> tasks` desaparece

3.4. BinProject

Este tipo de registro se utilizará para leer y escribir información de un proyecto en el fichero binario:

```
struct BinProject{
  char name[KMAXNAME];
  char description[KMAXDESC];
  unsigned numLists;
};
```

Los cambios que se han introducido con respecto a `Project` son los siguientes:

- Desaparece el campo `id`
- Como en `BinTask` y `BinList`, el campo `string name` se sustituye por el campo `char name[KMAXNAME]`
- El campo `string description` se sustituye por el campo `char description[KMAXDESC]`, donde `KMAXDESC` es una constante de tipo entero cuyo valor será 40
- Aparece un campo `unsigned numLists` que indica el número de listas que contiene el proyecto
- El campo `vector<List> lists` desaparece

3.5. BinToDo

Este tipo de registro se utilizará para leer y escribir información de toda la lista de proyectos del programa en el fichero binario:

```
struct BinToDo{
  char name[KMAXNAME];
  unsigned numProjects;
};
```

Los cambios que se han introducido con respecto a `ToDo` son los siguientes:

- Como en otros registros, el campo `string name` se sustituye por el campo `char name[KMAXNAME]`
- Aparece un campo `unsigned numProjects` que indica el número de proyectos que contiene el programa
- El campo `int nextId` desaparece
- El campo `vector<Project> projects` desaparece

Nota: el avezado lector se habrá dado cuenta de que los registros `BinList` y `BinToDo` son iguales en memoria (los campos que contienen son del mismo tipo), por lo que se podría haber utilizado un único tipo de registro para ambos. Sin embargo, para que el código sea más legible por un humano es mejor definir dos tipos de registros diferentes, como se ha hecho aquí.

4. Menú

Para esta práctica vas a tener que crear un nuevo menú principal como el que se muestra a continuación:

Terminal

```
1- Project menu
2- Add project
3- Delete project
4- Import projects
5- Export projects
6- Load data
7- Save data
8- Summary
q- Quit
Option:
```

Este menú será el que aparezca por pantalla cuando el usuario inicie el programa y por lo tanto deberá de gestionarse en la función `main`. El menú de la *Práctica 1* se utilizará para editar los datos de un proyecto en concreto, mostrándose cuando el usuario elija la opción `Project menu`, tal y como se describirá más adelante.

Las opciones válidas que puede introducir el usuario son los números del `1` al `8` y la letra `q`. Al igual que sucedía en la *Práctica 1*, si la opción elegida no es ninguna de éstas se emitirá el error `ERR_OPTION` llamando a la función `error` con dicho parámetro. Cuando el usuario elige una opción correcta, se debe ejecutar el código asociado a dicha opción. Al finalizarla, volverá a mostrar el menú principal y a pedir otra opción, hasta que el usuario decida salir del programa utilizando la opción `q`.

5. Opciones

En los siguientes apartados se describe el funcionamiento que deberá tener cada una de las ocho opciones que se muestran en el menú principal del programa.

5.1. Project menu

Esta opción permite editar los datos de un proyecto concreto (recuerda que en esta práctica podemos tener múltiples proyectos) usando para ello el menú de la *Práctica 1* y el código que lo gestiona. Esta función se activa cuando el usuario elige la opción `1` del menú principal. El programa pedirá en primer lugar que el usuario introduzca el identificador del proyecto que se quiere editar con el mensaje:

Terminal

```
Enter project id:
```

Si no existe ningún proyecto con ese identificador se emitirá el error `ERR_ID` y se volverá al menú principal. Si el identificador es correcto, se mostrará el menú de la *Práctica 1*, con la única diferencia de que la opción `q` se sustituirá por la opción `b`:

```
Terminal
1- Edit project
2- Add list
3- Delete list
4- Add task
5- Delete task
6- Toggle task
7- Report
b- Back to main menu
Option:
```

Esta opción `b` hará que el programa vuelva al menú principal. Todas las acciones que se realicen en este menú afectarán únicamente al proyecto con el identificador introducido. Su funcionamiento será exactamente el mismo que se definió en la *Práctica 1*.

- El código que tenías en la función `main` de la *Práctica 1* para manejar este menú deberás de moverlo a una nueva función, ya que en el `main` de esta nueva práctica lo que tendrás es la gestión del nuevo menú principal que se indicó al principio de este apartado
- Aunque no pueden haber listas con el nombre repetido dentro de un proyecto, sí que pueden existir listas con el nombre repetido en distintos proyectos

5.2. Add project

Esta opción permite añadir un proyecto nuevo al programa. Se activa cuando el usuario elige la opción `2` del menú principal. Para ello se pedirá al usuario que introduzca el nombre del proyecto mostrando el siguiente mensaje:

```
Terminal
Enter project name:
```

Si se introduce una cadena vacía, se emitirá el error `ERR_EMPTY` y se volverá a pedir al usuario que introduzca el nombre mostrando el mismo mensaje. Si ya existe en el programa un proyecto con ese nombre, se emitirá el error `ERR_PROJECT_NAME` y se volverá al menú principal.

Si el nombre es correcto, se asignará al proyecto y se pedirá a continuación al usuario que introduzca la descripción del mismo mostrando el siguiente mensaje:

```
Terminal
Enter project description:
```

A diferencia del nombre, la descripción puede quedarse vacía. Tras almacenar la nueva descripción del proyecto se asignará de manera automática al nuevo proyecto un identificador único que se almacenará en el campo `id` de `Project`. Este identificador vendrá dado por el valor que haya en ese momento almacenado en el campo `nextId` de la estructura `ToDo` que contiene toda la información del programa. Como se indicó en la Sección 3.1, este identificador único comenzará con el valor `1` y se incrementará de uno en uno para cada nuevo proyecto. Por lo tanto, cada vez que añadas un proyecto, deberás de incrementar el campo `nextId` para que al siguiente proyecto se le asigne un nuevo identificador. Tras asignar el identificador, se incorporará el nuevo proyecto al vector `projects` del registro `ToDo` del programa y se volverá al menú principal.

- Si se borra un proyecto no se reutilizará su identificador, ni aunque sea el último proyecto creado. Es decir, si tienes tres proyectos, con identificadores `1`, `2` y `3`, y se borra el `3`, al crear un nuevo proyecto se le asignará el identificador `4` aunque ya no exista un proyecto con el identificador `3`

5.3. Delete project

Esta opción permite borrar un proyecto existente en el programa. Se activa cuando el usuario elige la opción `3` del menú principal. Para ello se pedirá el identificador del proyecto con el mensaje:

Terminal
```
Enter project id:
```

Si no existe un proyecto con ese identificador, se emitirá el error `ERR_ID` y se volverá al menú principal. Si el identificador es correcto, se borrará el proyecto del vector `projects` del registro `ToDo` que almacena todos los proyectos del programa.

5.4. Import projects

Esta opción permite importar información de proyectos almacenados en un fichero de texto. Se activa cuando el usuario elige la opción `4` del menú principal. En primer lugar, se pedirá al usuario el nombre del fichero donde está almacenada la información mostrando el mensaje:

Terminal
```
Enter filename:
```

A continuación se abrirá el fichero de texto y se leerá, añadiendo al programa los proyectos almacenados en el fichero. Si no se puede abrir el fichero, se emitirá el error `ERR_FILE` y se volverá al menú principal. En un fichero puede haber datos de varios proyectos. Cada una de ellos tendrá el formato indicado en el siguiente ejemplo:

data.txt
```
<
#My first videogame
*My own project for developing an awesome videogame in C/C++
@Design
Sprites|12/4/2021|F|30
Backgrounds|10/6/2021|F|45
Logo|1/3/2021|T|12
@Code
Write code|1/8/2021|F|120
Unit tests|29/8/2021|F|60
|2/9/2021|F|50
Design classes/UML|28/2/2021|T|50
@Documentation, demo and more
Write user manual|1/10/2021|F|30
Prepare demo inputs|15/10/2021|F|45
Record demo|15/11/2021|F|35
Publish game (maybe in GitHub)|22/12/2021|F|40
>
<
#Programming 2
@Assignment 1
...
>
```

Este ejemplo muestra dos proyectos, "`My first videogame`" y "`Programming 2`". Todos los datos de un proyecto estarán contenidos entre el carácter `'<'` y el carácter `'>'`, que indican el comienzo y final de los datos del proyecto respectivamente. Estos dos caracteres aparecerán siempre solos en la línea, tal y como muestra el ejemplo.

Para cada proyecto, su nombre aparece en primer lugar precedido del carácter `'#'`. A continuación viene la descripción del proyecto precedida del carácter `'*'`. En el proyecto `Programming 2` no aparece la descripción, por lo que ese proyecto deberá crearse con la descripción vacía (esto es perfectamente válido, tal y como se vio en la *Práctica 1*). Después aparecen cada una de las listas de tareas con su nombre precedido del carácter `'@'`. Las tareas aparecen luego, cada una en una línea diferente. El carácter `'|'` separa los distintos campos de información: nombre de la tarea, fecha (con el formato `día/mes/año`), el carácter `'T'` si la tarea está completada o `'F'` si no lo está, y finalmente su duración en minutos.

Los proyectos que se lean del fichero se irán añadiendo al final del vector `projects` del registro `ToDo` en el mismo orden en el que aparezcan en el fichero, asignándoles un identificador único a cada uno (como si se hubieran creado mediante la opción `Add project`).

- Si al leer un proyecto de fichero ya existiera otro con el mismo nombre no se añadirá al vector, se emitirá el error `ERR_PROJECT_NAME` y se ignorarán todos sus datos, listas y tareas, pero se seguirá leyendo el fichero procesando la información de los siguientes proyectos si los hubiera
- Como sabes, no pueden existir en un proyecto dos listas con el mismo nombre. En este caso no hace falta que hagas esta comprobación, ya que nunca vamos a utilizar en la corrección un fichero de entrada que contenga dos listas con el mismo nombre para un mismo proyecto
- Lo que sí debes comprobar es que los datos de cada tarea sean correctos. Concretamente, deberás comprobar que la fecha introducida sea correcta y que el tiempo estimado de finalización también, siguiendo los mismos criterios que se establecieron en la *Práctica 1* para la función `Add task`. Si alguno de estos campos es incorrecto, se emitirá el error correspondiente (`ERR_DATE` o `ERR_TIME`, el que primero ocurra, pero nunca se mostrarán los dos) y no se introducirá la tarea en el vector `tasks` de la lista, pero se seguirá leyendo el fichero para procesar la siguiente línea
- La descripción de un proyecto puede estar vacía. En ese caso no aparecerá la línea que comienza con `'*'`
- Un proyecto puede no tener listas, con lo cual no aparecería ninguna línea con `'@'`
- Una lista puede no contener ninguna tarea
- El nombre de una tarea puede estar vacío. En el ejemplo de arriba, la tarea "`|2/9/2021|F|50`" tiene el nombre vacío, por lo que directamente aparece el carácter `'|'` en primer lugar
- El formato de las líneas del fichero será siempre correcto. Nunca se usarán los caracteres `'#'`, `'*'`, `'@'` o `'|'` dentro de nombres de proyecto, listas, tareas o descripciones
- El fichero de entrada puede estar vacío. En ese caso, no se importará nada, pero tampoco se mostrará ningún tipo de error

5.5. Export projects

Esta funcionalidad, que se activa con la opción `5` del menú principal, guardará en un fichero de texto información sobre los proyectos almacenados en el programa: nombre, listas y tareas asociadas. El formato en el que se guardará la información será el mismo que se ha descrito en la sección anterior para la importación de proyectos.

En primer lugar, se le preguntará al usuario si quiere guardar en el fichero todos los proyectos o uno en concreto, mostrando el siguiente mensaje:

Terminal

```
Save all projects [Y/N]?:
```

Si el usuario contestara con algo diferente a `Y/y` o `N/n`, se volvería a mostrar el mensaje anterior. Si el usuario contesta `Y` o `y`, se guardarán todos los elementos del vector `projects` del registro `ToDo` en el fichero, tal como se describe más abajo. Si contesta `N` o `n`, se le pedirá que introduzca el identificador del proyecto concreto que quiere guardar con el siguiente mensaje:

```
Terminal
Enter project id:
```

Como en otras ocasiones, si no existe un proyecto con ese identificador, se emitirá el error `ERR_ID` y se volverá al menú principal sin hacer ninguna operación sobre el fichero. Si todo es correcto, se pedirá al usuario el nombre del fichero donde desea almacenar la información, mostrando el siguiente mensaje:

```
Terminal
Enter filename:
```

Se abrirá el fichero de texto para escritura y se guardará en él la información correspondiente, volviendo a continuación al menú principal. Si no se puede abrir el fichero se emitirá el error `ERR_FILE` y se volverá al menú principal.

- Si el fichero de salida ya existe, se sobreescribirá todo su contenido borrando lo que hubiera antes almacenado en él

5.6. Load data

Esta opción permite cargar en el programa todos los datos de los proyectos a partir de un fichero binario. Los datos que hubiera en ese momento en la estructura `ToDo` del programa se borrarán completamente y se sustituirán por la información leída del fichero. Esta funcionalidad se activa con la opción 6 del menú.

En primer lugar, se pedirá al usuario el nombre del fichero binario donde está almacenada la información mostrando el mensaje:

```
Terminal
Enter filename:
```

Si el fichero no se pudiera abrir se emitirá el error `ERR_FILE`. Antes de proceder a la lectura del fichero, se solicitará confirmación al usuario mostrando el mensaje:

```
Terminal
Confirm [Y/N]?:
```

Si el usuario introduce `Y` o `y`, se procederá con el borrado de los datos actuales de la lista y con la lectura de los nuevos datos del fichero. Si el usuario introduce `N` o `n`, se volverá al menú principal sin llevar a cabo ninguna acción (excepto cerrar el fichero abierto). En caso de que el usuario introduzca cualquier otro carácter, se volverá a mostrar el mismo mensaje solicitando la confirmación.

La información almacenada en el fichero binario tendrá la siguiente estructura:

- En primer lugar aparecerá un registro de tipo `BinToDo` con el nombre (`name`) y número de proyectos guardados (`numProjects`)
- A continuación aparecerán tantos registros de tipo `BinProject` como indique el valor `numProjects`. Cada registro contendrá el nombre del proyecto (`name`), su descripción (`description`) y el número de listas (`numLists`). Los registros `BinProject` no van justo uno a continuación del otro en el fichero, sino que cada registro `BinProject` lleva justo detrás toda la información que se indica en los siguientes dos puntos

- Detrás de cada registro de tipo `BinProject` aparecerán `numLists` registros de tipo `BinList`, con el nombre de la lista (`name`) y el número de tareas (`numTasks`)
- Después de cada registro de tipo `BinList` aparecerán `numTasks` registros de tipo `BinTask`, cada uno de los cuales tendrá el nombre de la tarea (`name`), la fecha de finalización (`deadline`), si está completado o no (`isDone`) y el tiempo estimado (`time`)

La siguiente imagen muestra un ejemplo de cómo podría estar estructurado un fichero binario (es solo un dibujo para aclara la estructura; recuerda que el fichero en realidad solo contiene ceros y unos):

BinToDo	BinProject	BinList	BinTask	BinTask	BinProject	BinList	BinTask
name	name	name	name	name	name	name	name
My ToDo list	Programming 2	Assignment 1	Test	Report	My videogame	Sprites	Design
numProjects	description	numTasks	deadline	deadline	description	numTasks	deadline
2	Best subject ever	2	12/2/2021	4/3/2021	GTA VI	1	11/8/2022
	numLists		isDone	isDone	numLists		isDone
	1		True	False	1		False
			time	time			time
			50	150			95

En primer lugar aparecería en el fichero un registro de tipo `BinToDo`. Este registro indica que tenemos dos proyectos almacenados en el fichero (`numProjects` vale 2). A continuación aparece un registro `BinProject` representando el primero de estos proyectos, cuyo nombre (`name`) es `Programming 2`. Este registro tiene una lista asociada (`numLists` vale 1). A continuación aparece esta lista, representada por una estructura de tipo `BinList` cuyo nombre (`name`) es `Assignment 1`. Esta lista tiene dos tareas (`numTasks` vale 2). Detrás de esta estructura aparecen los dos registros de tipo `BinTask` que representan esas dos tareas (llamadas `Test` y `Report`). Con esto ya tenemos toda la información del primer proyecto. A continuación aparece un nuevo registro de tipo `BinProject` que representa el segundo proyecto almacenado en el fichero (con el nombre `My videogame`). Este proyecto contiene una única lista (`numLists` vale 1), cuyo registro `BinList` aparece justo a continuación. Esta lista tiene asociada una única tarea (`numTasks` vale 1), la cual aparece después en un registro de tipo `BinTask` (con el nombre `Design`). Con este último registro se completa toda la información almacenada en el fichero.

- Se asume que todos los datos almacenados en el fichero binario son correctos, por lo que no es necesario hacer ninguna comprobación adicional. Por ejemplo, no habrá proyectos con el nombre repetido
- En el fichero binario no se almacena el identificador de los proyectos, por lo que al leer un proyecto hay que asignarle el siguiente identificador disponible (como si se hubiera añadido con la opción `Add project`)

5.7. Save data

Esta opción permite guardar todos los proyectos del programa en un fichero binario. Se activa con la opción 7 del menú. En primer lugar, se pedirá al usuario el nombre del fichero binario donde se almacenará la información, mostrando el mensaje:

```
Terminal
Enter filename:
```

Si el fichero no se pudiera abrir, se emitiría el error `ERR_FILE` y se volverá al menú principal. Si el fichero ya existe, se sobrescribirá todo su contenido borrando lo que hubiera antes almacenado en él. El formato de salida será exactamente el mismo que se ha descrito en el apartado anterior para la opción `Load data`.

- Ten en cuenta que puede que tengas que recortar los campos de texto (por ejemplo, `name` y `description` para `Project`) al guardarlos en el fichero binario. Por ejemplo, podrías tener un registro `Project` con una cadena de 50 caracteres almacenada en el campo `name`. Sin embargo, en el fichero binario tienes que guardar una estructura de tipo `BinProject` cuyo campo `name` tiene un tamaño máximo de 20. En este caso, tendrías que recortar el nombre original y almacenar solo los primeros 19 caracteres en fichero (el máximo admitido, ya que tienes que dejar un espacio para incluir el `'\0'` final)

5.8. Summary

Esta opción mostrará un resumen de los proyectos existentes en el programa. Se activará con la opción `8` del menú. Para cada proyecto se mostrará su identificador entre paréntesis, su nombre y la cantidad de tareas realizadas sobre el total (de entre todas las listas que contenga) entre corchetes. Un posible ejemplo de salida sería ésta:

```
(1) My first videogame [2/11]
(3) Programming 2 [0/8]
```

Este ejemplo muestra dos proyectos. El primero de ellos tiene el identificador `1`, el nombre "`My first videogame`" y tiene `2` tareas realizadas de las `11` que contiene en total entre todas sus listas. El segundo tiene el identificador `3`, el nombre "`Programming 2`" y tiene `0` tareas realizadas de las `8` que contiene en total entre todas sus listas.

6. Argumentos del programa

En esta práctica el programa debe permitir al usuario indicar algunas acciones a realizar mediante el paso de argumentos por línea de comando. Deberás utilizar los parámetros `int argc` y `char *argv[]` de la función `main` para poder gestionarlos. El programa admitirá los siguientes argumentos por línea de comando:

- `-i <fichero texto>`. Permite importar proyectos de un fichero de texto, cuyo nombre deberá indicarse tras el argumento `-i`, comportándose igual que si el usuario seleccionara la opción `Import projects` del menú principal, también a la hora de tratar los errores si alguno de los datos leídos es incorrecto. Por ejemplo, si se lee del fichero un proyecto cuyo nombre ya existe, se emitirá el error `ERR_PROJECT_NAME`, se ignorarán los datos de ese proyecto y se seguirá procesando el resto del fichero. Si no se puede abrir el fichero se emitirá el error `ERR_FILE` y se finalizará el programa sin llegar a mostrar el menú principal. El siguiente ejemplo importará los datos almacenados en el fichero de texto `data.txt`:

  Terminal
  ```
  $ prac2 -i data.txt
  ```

- `-l <fichero binario>`. Permite cargar información de los proyectos desde un fichero binario cuyo nombre deberá indicarse después del argumento `-l`, comportándose igual que si el usuario seleccionara la opción `Load data` del menú. En este caso no deberá pedirse confirmación al usuario para cargar los datos desde el fichero. Si no se puede abrir el fichero, se emitirá el error `ERR_FILE` y se finalizará el programa sin llegar a mostrar el menú principal. El siguiente ejemplo cargará los datos almacenados en el fichero binario `data.bin`:

  Terminal
  ```
  $ prac2 -l data.bin
  ```

- Los dos argumentos son opcionales y pueden no aparecer en la llamada al programa
- Un determinado argumento solo puede aparecer una vez en la llamada, de lo contrario se considerará que los argumentos son erróneos
- Los ficheros binarios no tienen por qué tener necesariamente la extensión `.bin` y los de texto no tienen por qué tener la extensión `.txt`. Utilizamos estas extensiones en los ejemplos para que quede más claro con qué tipo de fichero estamos trabajando

En caso de que se produzca un error en los argumentos de entrada (por ejemplo, que se introduzca una opción que no existe), se deberá emitir el error `ERR_ARGS` y se finalizará el programa sin llegar a mostrar el menú principal. Si todos los argumentos son correctos, se procesarán las opciones introducidas por el usuario y posteriormente se mostrará el menú principal de programa de la forma habitual. Los argumentos pueden ser erróneos si se da alguna de las siguientes circunstancias:

- Alguna de las opciones aparece repetida. Por ejemplo:

```
Terminal
$ prac2 -l data.bin -l moredata.bin
```

- A las opciones `-l` o `-i` les falta el argumento de detrás. Por ejemplo:

```
Terminal
$ prac2 -l data.bin -i
```

- Aparece un argumento que no es ninguno de los permitidos. Por ejemplo:

```
Terminal
$ prac2 -n data.txt
```

Los dos argumentos pueden aparecer al mismo tiempo en una llamada al programa y además en cualquier orden. Independientemente del orden en el que aparezcan en la llamada, el orden en el que se procesarán los argumentos será el siguiente:

1. En primer lugar se procesará la opción `-l` para cargar datos de un fichero binario
2. En segundo lugar se ejecutará la opción `-i` para importar datos desde un fichero de texto

En el siguiente ejemplo, en primer lugar se cargarán los datos del programa almacenados en el fichero `data.bin` y a continuación se incorporarán los proyectos almacenados en el fichero `data.txt`:

```
Terminal
$ prac2 -i data.txt -l data.bin
```

Otro ejemplo, cuyos parámetros son válidos, sería el siguiente:

```
Terminal
$ prac2 -l -l -i -t
```

Aunque a primera vista los parámetros pudieran parecer incorrectos, en realidad no lo son. El programa recibe un primer parámetro `-l` que va seguido de un fichero que se llama `-l`. A continuación recibe un parámetro `-i` que va seguido de un fichero que se llama `-t`. Esta secuencia de parámetros es, por tanto, correcta.