Fundamentos de Programación
PR 4 - 20232

Fecha límite de entrega: **19/06/2024 a las 23:59**

Presentación

En esta práctica se presenta un caso que hemos trabajado durante las PR previas de este semestre, es decir, en las PR1, PR2 y PR3. El contexto, pues, será el mismo, pero ampliándolo con la gestión de las naves en las bases estelares y su asignación posterior a las misiones que emprende el Imperio.

Se quiere, también, que continuéis poniendo en juego una competencia básica en la programación: la capacidad de entender un código ya dado y saber adaptarlo a las necesidades de nuestro problema. Con esta finalidad, se os facilita gran parte del código, en el que existen acciones y funciones muy similares a las que se piden. Se trata de aprender mediante ejemplos, una habilidad muy importante que debéis desarrollar.

Descripción del proyecto

Este semestre nos han pedido crear una aplicación para gestionar las naves del Imperio Galáctico y su distribución en bases estelares por la galaxia. Hasta ahora, en las PR anteriores, hemos ido desarrollando operaciones para tratar estos datos, enfocadas sobre todo en las naves. En la PR4, ampliaremos las operaciones para gestionar la ubicación de las naves en las bases estelares y su asignación a las misiones que se han de abastecer. En el código que os proporcionamos como base para la realización de esta práctica, podréis encontrar la estructura para almacenar todos estos datos y algunos de los algoritmos trabajados hasta el momento.

Las acciones que querremos tener programadas serán las siguientes:

- Leer, mediante un menú interactivo, los datos de una base estelar.
- Cargar los datos desde archivos y también almacenarlos.
- Realizar búsquedas, aplicar filtros y obtener datos estadísticos.
- Gestionar la distribución de las naves en las misiones, atendiendo a las necesidades de la misión.

Además, aunque esta primera versión será una aplicación online de control de recursos, queremos que todas estas funcionalidades queden recogidas en una **API** (*Application Programming Interface*), lo que nos permitirá en un futuro poder utilizar esta aplicación en diferentes dispositivos (interfaces gráficas, teléfonos móviles, tabletas, web...).

Estructuración del código

Junto con el enunciado se os facilita un proyecto CodeLite que será el esqueleto de la solución de su práctica. En la práctica debéis trabajar con este código, al que no será necesario añadir ningún archivo más. Únicamente completaréis los tipos, funciones y/o acciones que se os vayan indicando en el enunciado. A continuación se dan algunas indicaciones del código proporcionado:

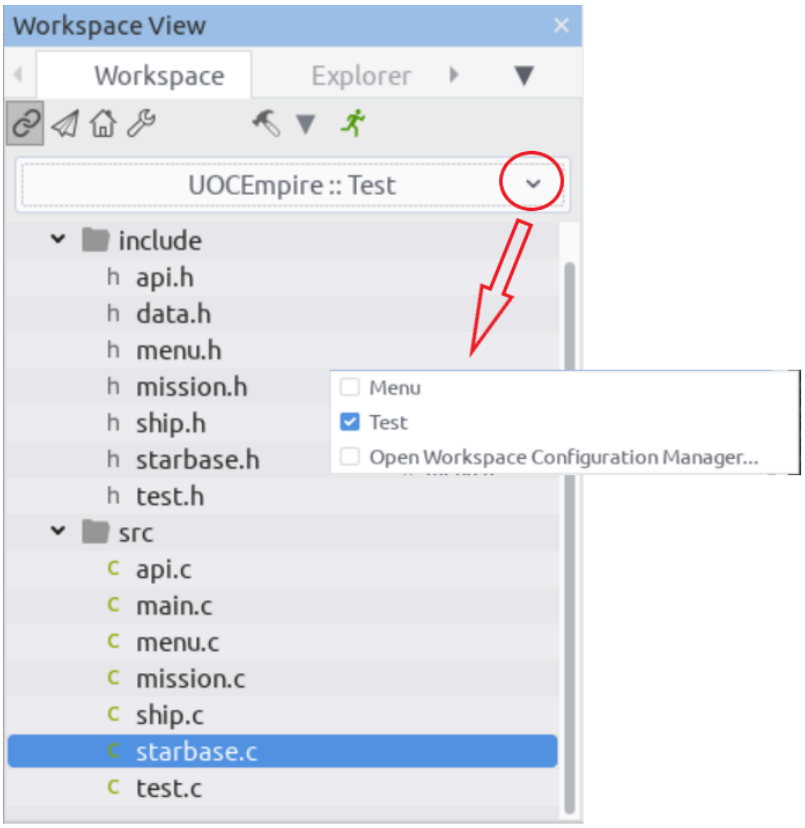
main.c

Contiene el inicio del programa. Está preparado para funcionar en dos modos distintos, en modo *Menú y* en modo *Test*. Para gestionar qué modo debe ejecutar la aplicación se utilizan los parámetros pasados en el programa desde el entorno de ejecución. Estos parámetros se corresponden con los parámetros definidos en la función principal*:*

```
int main(int argc, char **argv)
```

Para que funcione en modo *Menú* no es necesario pasar ningún parámetro a la aplicación y, para que funcione en modo *Test*, es necesario pasar el parámetro "-t" a la aplicación.

Este funcionamiento ya está implementado y el proyecto de *Codelite* se ha configurado con dos modos de configuración: *Menú* y *Test*. Podéis cambiar el modo de ejecución utilizando el desplegable *Configuration Manager* del editor *CodeLite:*



- La aplicación en **modo Menú** nos muestra por el canal de salida estándar un menú que permite al usuario interactuar con la aplicación: realizar altas y bajas de naves, bases estelares, aprovisionamiento de las misiones, etc. También obtener información, estadísticas y otras funcionalidades relacionadas con los datos de la aplicación.
- La aplicación en **modo Test** ejecuta un conjunto de pruebas sobre el código para asegurar que todo funciona correctamente. El resultado de estas pruebas se mostrará una vez finalizada la ejecución por el canal de salida estándar.

  Inicialmente, muchas de estas pruebas fallarán, pero, una vez realizados todos los ejercicios, todas deberían pasar correctamente. Es importante que os **aseguréis que vuestro programa, una vez completados todos los ejercicios, pasa la totalidad de las pruebas incluidas en el proyecto**. Una vez superados el conjunto de pruebas del **modo Test** en local, **tendréis que subir a DSLab únicamente los archivos que se pide que modifiquéis y pasar el juego de pruebas de la PR4**.

data.h

Se definen los tipos de datos que se usan en la aplicación. Aunque se podrían separar en diferentes archivos de cabecera, se han agrupado todos para facilitar la lectura del código.

starbase.h / starbase.c

Contienen todo el código que gestiona las bases estelares, con su estructuración en niveles y hangares.

ship.h / ship.c

Contienen el código que gestiona las naves.

mission.h / mission.c

Contienen el código que gestiona los recursos necesarios para las misiones.

menu.h / menu.c

Contienen todo el código para gestionar el menú de opciones que aparece cuando se ejecuta en modo Menú.

api.h / api.c

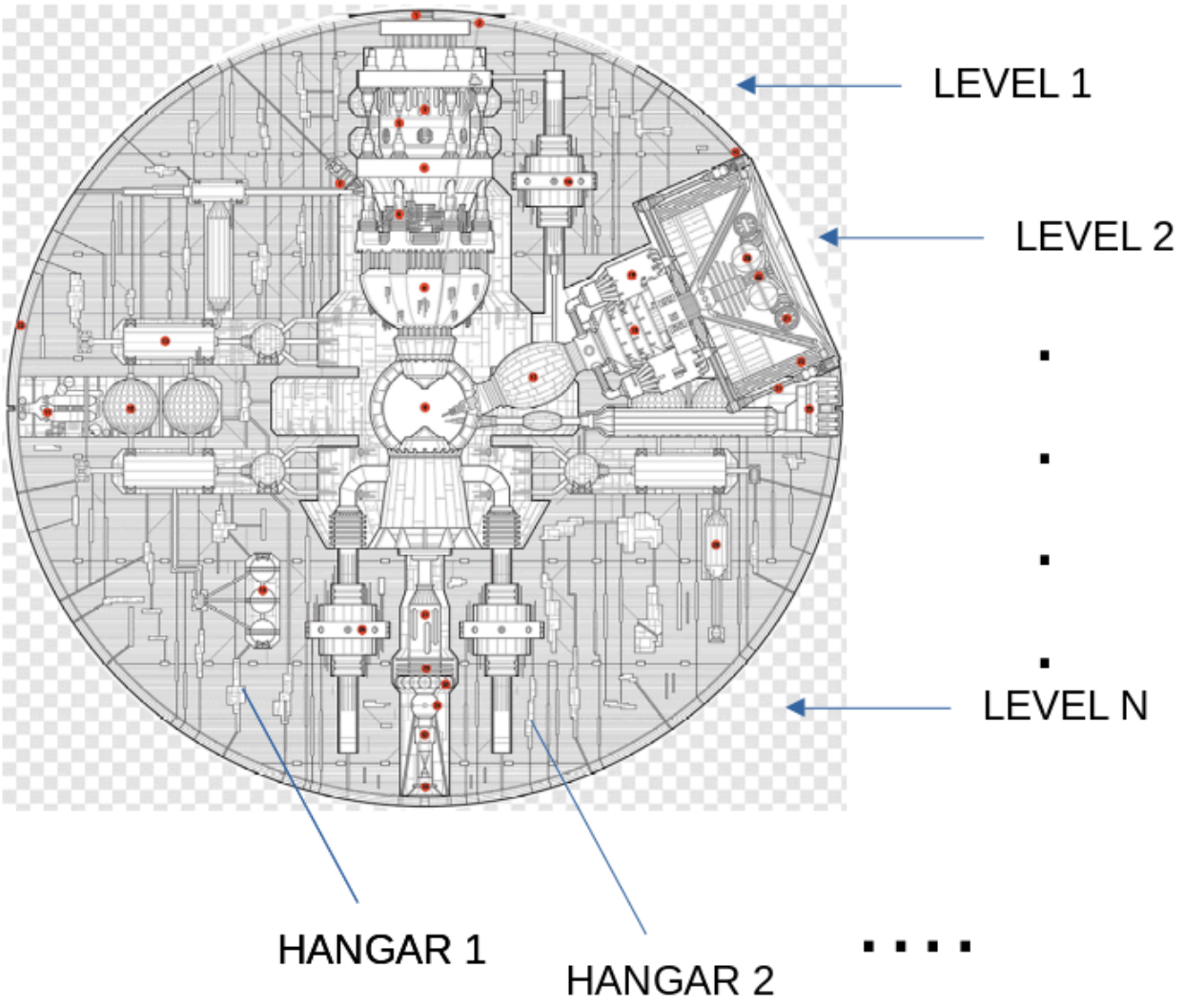
Contienen los métodos (acciones y funciones) públicos de la aplicación, lo que sería la API de nuestra aplicación. Estos métodos son los que se llaman desde el menú de opciones y los que invocaría cualquier otra aplicación que quisiera utilizar el código.

test.h / test.c

Contienen todas las pruebas que se pasan en el código cuando se ejecuta en modo Test.

Enunciado

Hemos realizado una toma de requisitos con la organización *UOCEmpire* y nos han detallado cómo se estructura una base estelar, desde el punto de vista de almacenamiento de recursos. Considerad el siguiente esquema:



En esta sección transversal de la **base estelar** (**starbase**) veréis que la base se organiza en **niveles** (**levels**), comenzando desde la parte superior de la base hasta la inferior. Y, dentro de cada nivel, encontraremos **hangares** (**hangars**) dedicados a almacenar recursos de un tipo específico. Los habrá que guarden suministros (STORAGE), los que albergan las naves asignadas en la base estelar (SHIPS) y otros de propósito general (OTHERS). En el caso de los dedicados a naves, se diseñan de manera que contengan el mismo tipo de nave y así se puede optimizar el espacio disponible.

Con toda la información recogida, hemos identificado los campos que son necesarios en la **estructura de datos de las bases estelares**. El conjunto de campos de la tupla **tStarbase** queda resumida en la siguiente tabla.

Campo	Descripción	Tipo / validación
id	Identificador de la base.	De tipo ***tStarbaseId***.
baseName	Nombre de la base.	**Cadena** de, como máximo, 15 caracteres alfanuméricos y sin espacios. Si hay necesidad de guardar un nombre compuesto, utilizaremos el guion bajo ('_') como separador de las palabras.
planet	Nombre del planeta en cuya órbita se encuentra la base.	**Cadena** de, como máximo, 20 caracteres alfanuméricos y sin espacios. Si hay necesidad de guardar un nombre compuesto, utilizaremos el guion bajo ('_') como separador de las palabras.
sector	Identificador del sector de la galaxia en que se encuentra la base.	De tipo ***tSectorId***.
levels	Tabla de niveles en que se estructura la base.	Tabla ***tLevelsTable*** que puede contener, como máximo, 10 niveles. Estos niveles se representarán con el tipo ***tLevel***.

Como vemos en los campos de la tabla anterior, una base estelar se divide, en niveles que representaremos con el tipo **tLevel**:

Campo	Descripción	Tipo / validación
id	Identificador del nivel	De tipos ***tLevelId***.
hangars	Tabla de los hangares que puede haber en un nivel.	Tabla ***tHangarsTable*** que contiene hasta un máximo de 20 hangares (***tHangar***).

Cada hangar se representa con el tipo de datos **tHangar**, que consta de los siguientes campos:

Campo	Descripción	Tipo / validación
id	Identificador del hangar.	De tipo ***tHangarId***.
isAvailable	Indica si el hangar está operativo o ha sufrido algún desperfecto que lo haga inutilizable.	**Booleano**
hangarType	Tipo de recurso que puede almacenar el hangar.	Tipo enumerado ***tHangarType***, que admite tres tipos de recursos: STORAGE, SHIPS y OTHERS.
hangarShipType	Tipo de nave que se alojará en el hangar.	Tipo enumerado ***tShipType***, que puede ser de cinco clases: CARRIER, TRANSPORT, FIGHTER, MEDICAL y EXPLORER.
nShips	Número de naves que hay alojadas en el hangar.	Entero. Tendrá valor = 0 si el hangar no está dedicado a almacenar naves.
shipsId	Identificadores de las naves que están en el hangar actualmente.	Vector de un máximo de 30 posiciones que guardarán los identificadores (***tShipId***) de las naves.

Aparte de la estructura que representa las bases estelares, necesitaremos otra estructura donde podamos representar las misiones que el Imperio necesita aprovisionar de naves. Vemos la representación de **tMission**:

Campo	Descripción	Tipo / validación
id	Identificador de la misión.	De tipo ***tMissionId***.
startStarbase	Identificador de la base estelar donde se iniciará el aprovisionamiento de naves.	De tipo ***tStarbaseId***.
shipsNeed	Número de naves, en total, que se necesita para realizar la misión.	**Entero**.

shipsTypes	Número de naves que se necesitan de cada tipo para realizar la misión.	Vector de 5 posiciones donde se guardará, en cada una, el número de naves **(entero)** que se necesitan de cada tipo. Se respeta el orden de los tipos de naves definidos: CARRIER, TRANSPORT, FIGHTER, MEDICAL y EXPLORER.
assignedShips	Número de naves que ya han sido asignadas a la misión.	**Entero**. Valor máximo = 20, que son las posiciones que podrá tener el vector de naves asignadas (***assignedShipsInfo***)
assignedShipsInfo	Información de las naves asignadas a la misión.	**Vector** de un máximo de 20 posiciones donde guardaremos elementos del tipo **tAssignedShip**. Este tipo tendrá la información correspondiente al identificador de la nave y a los identificadores de la base estelar, nivel y hangar donde estaba la nave antes de ser asignada a la misión.

La empresa *UOCEmpire* ya nos ha dado el visto bueno al análisis entregado, y podemos empezar a realizar los cambios. Para implementar estos cambios, tal y como ya hemos comentado, partiremos del código fuente que nos han proporcionado, e iremos añadiendo el código necesario.

Además de los tipos estructurados presentados, revisad la definición de constantes existentes en el código. Será necesario utilizar algunas de ellas para resolver los ejercicios.

Ejercicio 1: Definición de tipos [5%]

Se pide definir el tipo de dato ***tAssignedShip*,** en el archivo data.h, que nos servirá para guardar la información de la nave que ha sido asignada en una misión. En concreto, nos interesa almacenar el identificador de la nave, el identificador de la base estelar, así como el identificador del nivel y del hangar dentro de esa base, donde estaba la nave antes de asignarla a la misión.

NOTA:

Una vez hecho esto, **descomentad la definición de la variable `TYPEDEF_COMPLETED` que encontraréis al principio del archivo data.h** antes de continuar con el resto de ejercicios.

Ejercicio 2: Funciones de copia y comparación [10%]

Un problema que nos encontramos con los tipos estructurados es que muchos de los operadores que tenemos definidos con los tipos básicos de datos, como los de comparación (==, !=, <, >, ...) o el de asignación (=), no funcionan para los nuevos tipos que nos creamos.

Por eso, a menudo se hace necesario definir acciones o funciones que nos den estas funcionalidades. Por ejemplo, ya hemos visto que para asignar una cadena de caracteres no lo hacemos con el operador de asignación normal (=), sino que debemos recurrir a la función strcpy. Lo mismo ocurre en las comparaciones, donde en vez de utilizar los operadores normales (==, !=, <, >, ...) utilizamos **strcmp**.

Se pide:

a) [5%] Completad, en el archivo starbase.c, la acción ***hangarCpy*** que permite copiar todos los datos de una estructura *tHangar* a otra. Podéis ayudaros de la acción *levelCpy* para diseñarla.

b) [5%] Completad, en el archivo starbase.c, la función ***starbaseCmp*** que compara dos bases estelares, sb1 y sb2, y devuelve:

-1 si sb1<sb2, 0 si sb1=sb2, 1 si sb1>sb2

El orden de las bases estelares vendrá dado por sus campos, con el siguiente orden de prioridad, de más a menos prioritario:

- ocupación, i.e. relación porcentual entre el número de naves que hay en toda la base y el número máximo de naves que podría haber (descendente)
- nombre de la base estelar (ascendente)
- nombre del planeta donde se ubica (ascendente)
- sector de la galaxia donde se ubica (ascendente)
- número de niveles de la base (descendente)
- número de hangares operativos dentro de la base (descendente).

NOTA: Para probar este ejercicio podéis usar la opción *Manage starbases > Copy starbase*

Esto significa que, si sb1 es una base estelar con un 40% de ocupación y sb2 tiene un 50%, decidiremos que sb1 < sb2. En caso de que los porcentajes sean iguales, comprobaremos que el

nombre de la base sea anterior, alfabéticamente hablando (podéis usar la acción strcmpUpper, que os proporcionamos para comparar cadenas de caracteres). Si son iguales, miraremos el nombre del planeta. En caso de empate continuaremos con el sector, luego con el número de niveles y, finalmente, el número de hangares operativos de la base. Si todos los datos son iguales, significará que sb1=sb2.

Ejercicio 3: Operaciones básicas con tablas [10%]

Se pide disponer de unas acciones para comprobar si una nave se encuentra en alguna de las bases estelares del Imperio. Para ello:

a) [2,5%] Completad la acción ***isShipInHangar*** en api.c que, dado un *tHangar* y un identificador de nave *tShipId*, compruebe si la nave se encuentra en el hangar. Devolverá cierto si la encuentra y falso, en caso contrario.

b) [2,5%] Completad la acción ***isShipInLevel*** en api.c que, dado un *tLevel* y un identificador de nave *tShipId*, compruebe si la nave se encuentra en ese nivel. Es obligatorio utilizar la acción implementada en el apartado anterior que hace esta misma comprobación, pero a nivel de hangar.

c) [2,5%] Completad la acción ***isShipInStarbase*** en api.c que, dado una *tStarbase* y un identificador de nave *tShipId*, compruebe si la nave se encuentra en esa base estelar. Es obligatorio utilizar la acción implementada en el apartado anterior que hace esta misma comprobación, pero para un nivel de la base.

d) [2,5%] Completad la acción ***isShipInAnyStarbase*** en api.c que, dado una *tStarbaseTable* y un identificador de nave *tShipId*, compruebe si la nave se encuentra en alguna de las bases estelares del Imperio. Es obligatorio utilizar la acción implementada en el apartado anterior que hace esta misma comprobación, pero a nivel de base estelar.

Ejercicio 4: Entrada interactiva de datos [10%]

Completad la acción ***readHangar*** (menu.c) para que lea por teclado el tipo de naves que puede almacenar el hangar, cuántas se le han asignado y los identificadores de todas ellas. Tened presente que no se puede admitir asignar una nave, si los datos de dicha nave no se encuentran registrados en la estructura de datos que almacena las naves. Validad esta condición y, en caso de que no se cumpla, informad con este mensaje:

```
Ship unknown in ships table. Please, select another one.
```

NOTA: Para probar este ejercicio, usad la opción *Manage starbases > Starbase Add*

NOTA: Podéis usar las funciones del ejercicio 3 para validar la condición (api.c).

Ejercicio 5: Filtros y ordenaciones [25 %]

Se pide que implementéis las siguientes acciones relacionadas con filtros y ordenaciones:

a) [5%] Implementad la acción ***starbaseTableFilterBySector*** (starbase.c) que, a partir de una tabla de bases estelares, devuelva las que están ubicadas en el sector que se indica.

   NOTA: Para probar el ejercicio, usad la opción *Manage starbases > View sector starbases.*

   NOTA: Usad las acciones *starbaseTableInit* y *starbaseTableAdd* (starbase.c)

b) [10%] Implementad el código de la acción ***starbaseTableOrderByOccupation*** (starbase.c) que, a partir de una tabla de bases estelares, modifique esta misma tabla con las bases ordenadas de menor a mayor ocupación. La acción debe resolverse con un algoritmo de ordenación por selección (que se puede consultar en el apartado 6.2 de la xwiki).

   NOTA: Para las pruebas, usad *Manage starbases > Sort starbases by occupation.*

c) [10%] Implementad el código de la acción ***missionTableSortByShips*** (mission.c) que, a partir de una tabla de misiones, modifique esta misma tabla con las misiones ordenadas de mayor a menor según los parámetros de la misión sean mejores o peores. Usad la función de comparación *missionCmp* para saber si una misión es mejor que otra. La acción debe resolverse por el algoritmo de ordenación por selección (apartado 6.2 de la wiki).

Ejercicio 6: Procesamiento de las misiones [20%]

Cuando el Imperio Galáctico comienza una nueva campaña, los mandos generan una lista de misiones para llevar a cabo en los próximos meses. Para poder realizarlas, es necesario que se puedan aprovisionar las naves que participarán en cada misión.

Una misión comienza en una base estelar, indicada con el identificador de dicha base (*startStarbase*), donde selecciona el número de naves de cada tipo que se especifican en la misión. Por cada nave seleccionada, se eliminará su identificador del hangar de la base estelar donde estaba estacionada y se guardará en la misión los datos de dicha ubicación (identificador de la nave, de la base estelar, del nivel dentro de la base y del hangar).

Si no se pudiera abastecer el número de naves necesarias para la misión en la base estelar de inicio, se probaría con el resto de bases que están en el mismo sector que la inicial. Y, si finalmente no se pudiera abastecer por completo de naves la misión, se mostrará un mensaje por pantalla indicándolo.

```
No ships enough in sector. Mission cannot be processed
```

El proceso de aprovisionamiento de las naves se lleva a cabo en la acción ***processMIssion***. (api.c) **que se pide que implementéis,** siguiendo las pautas que se indican a continuación:

processMission:

- Primero **comprobaremos si la misión ha sido procesada** ya. Si es así, ya tendrá asignado el número de naves requeridas para la misión y finalizará la acción.
- Obtendremos de la información de la misión el **número de naves de cada tipo** que se necesitan aprovisionar.
- Para cada tipo de nave requerido, buscaremos en todos los hangares de la base para ver si hay de ese tipo. Si encontramos una nave del tipo correcto, **se asignará a la misión y se eliminará del hangar de la base estelar**.
- Si hemos mirado en todos los hangares y aún nos faltan naves por aprovisionar, obtendremos la **lista de bases estelares** que hay en el **mismo sector** de la galaxia y repetiremos el proceso para cada una de ellas.
- El proceso **finalizará** cuando se hayan **asignado todas las naves** requeridas **o** cuando ya **no queden bases estelares** en el sector para inspeccionar.

  NOTA: Para probar el ejercicio, usad *Manage missions > Process Mission.*

  NOTA: Podéis apoyaros en la acción *starbaseTableFilterBySector* (api.c) diseñada en el ejercicio 5a para obtener las bases estelares de un sector.

  NOTA: Podéis usar la función auxiliar *starbaseNumberShipsType* (api.c) que, dada una base estelar y un tipo de nave, devuelve el número de naves de ese tipo que hay en la base. También podéis usar la acción auxiliar *assignShipsToMission* (api.c) que desasigna un número de naves de un tipo dado de una base estelar y las asigna a una misión.

Ejercicio 7: Escritura de datos por pantalla [5%]

Cuando ya se han procesado las misiones, los encargados de trasladar las naves para cada misión necesitan un listado impreso con los detalles específicos de cada una. Por tanto, se pide que implementéis la acción ***printMissionInfo*** (menu.c) que se encargará de **mostrar por pantalla los datos de la misión y de las naves que tiene asignadas**. El formato de salida de la información será el siguiente:

1. El identificador de la misión.
2. Los datos de la base estelar inicial (identificador, nombre y planeta de ubicación).
3. La cantidad de naves que se requieren de cada tipo.
4. Los datos de las naves asignadas en la misión (identificador de la nave, nombre de la base estelar donde estaba ubicada, así como el identificador del nivel y hangar en que se encontraba)

Aquí tenéis un ejemplo de cómo se verían los datos de una misión:

```
-----------------------------------------------------
  Mission identifier: 3
-----------------------------------------------------
  Mission starbase initial....
  Identifier: 1
  Name: DEATH_STAR
  Planet: ALDERAAN
-----------------------------------------------------
```

```
  Mission resources....
  Ships CARRIER type: 1
  Ships TRANSPORT type: 1
  Ships FIGHTER type: 1
  Ships MEDICAL type: 1
  Ships EXPLORER type: 1
-----------------------------------------------------
  Assigned ships....
  [1] Id: 4 Starbase: DEATH_STAR Level: 16 Hangar: 163
  [2] Id: 9 Starbase: DEATH_STAR Level: 19 Hangar: 194
  [3] Id: 14 Starbase: DEATH_STAR Level: 14 Hangar: 141
  [4] Id: 21 Starbase: DEATH_STAR Level: 14 Hangar: 144
  [5] Id: 26 Starbase: DEATH_STAR Level: 19 Hangar: 192
-----------------------------------------------------
```

Y, a continuación, la salida que se obtendría si se intenta mostrar la información de una misión que aún no tiene las naves necesarias asignadas:

```
-----------------------------------------------------
  Mission identifier: 4
-----------------------------------------------------
  Mission starbase initial....
  Identifier: 3
  Name: RESURGENT
  Planet: SCARIF
-----------------------------------------------------
  Mission resources....
  Ships CARRIER type: 0
  Ships TRANSPORT type: 2
  Ships FIGHTER type: 1
  Ships MEDICAL type: 1
  Ships EXPLORER type: 1
-----------------------------------------------------
  Ships: Not yet assigned
-----------------------------------------------------
```

NOTA: Para probar el ejercicio utilizad la opción de *Manage missions* > *Print detail mission*.

Ejercicio 8: Estadísticas [15%]

Se pide que implementéis tres acciones:

a) [5%] Implementad la acción ***starbaseTableFilterByOccupation*** (starbase.c) que, a partir de una tabla de bases estelares, devuelva aquellas que tienen un porcentaje de ocupación de naves por encima de un valor dado. Se entiende por porcentaje de ocupación la relación entre el número de naves que hay en total en la base estelar y el máximo de naves que podría haber en toda la base.

b) [5%] Implementad la acción ***starbaseTableAvgOccupation*** (starbase.c) que, a partir de una tabla de bases estelares, devuelva el porcentaje de ocupación promedio de todas las bases. Para resolver este ejercicio, acumularemos el número de naves que hay en todas las bases estelares actualmente y lo dividiremos entre el número máximo de naves que pueden albergar todas esas bases.

c) [5%] Implementad la acción ***shipTableSelectShips*** (ship.c) que, a partir de una tabla de naves, devuelve aquellas que sean de un tipo concreto pasado por parámetro.

NOTA: Para probar este ejercicio, podéis utilizar las opciones interactivas que encontraréis en el *menú principal > View statistics*