Práctica 2

Formato y fecha de entrega

Hay que entregar la Práctica antes del día **10 de junio a las 23:59h**. Para la entrega deberéis entregar un fichero en formato **ZIP** de nombre ***logincampus*_pr2** en minúsculas (donde *logincampus* es el nombre de usuario con el que hacéis *login* en Campus). El ZIP tiene que contener:

- Workspace CodeLite entero, con todos los ficheros que se piden.

Para reducir la medida de los ficheros y evitar problemas de envío que se pueden dar al incluir ejecutables, hay que eliminar el que genera el compilador. Podéis utilizar la opción "Clean" del workspace o eliminarlos directamente (las subcarpetas Menú y Test son las que contienen todos los binarios que genera el compilador).

Hay que hacer la entrega en el apartado de entregas de EC del aula de teoría.

Presentación

Esta práctica culmina el proyecto empezado a la práctica 1. Hemos construido una aplicación para gestionar los aviones y los pasajeros de una empresa de transporte aéreo y ahora nos piden gestionar los vuelos. Trabajaremos con un TAD cola para representar las colas de embarque que permitirán, a los diferentes pasajeros, acceder a los aviones que los tienen que llevar a sus destinos.

Competencias

Transversales

- Capacidad de comunicación en lengua extranjera.

Específicas

- Capacidad de diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y reutilización.
- Conocimientos básicos sobre el uso y la programación de los ordenadores, sistemas operativos, y programas informáticos con aplicación a la ingeniería.

Objetivos

- Analizar un enunciado y extraer los requerimientos tanto de tipos de datos como funcionales (algoritmos)
- Analizar, entender y modificar adecuadamente código existente
- Saber utilizar tipo de datos abstractos
- Saber utilizar punteros

Recursos

Para realizar esta actividad tenéis a vuestra disposición los siguientes recursos:

- Materiales en formato web de la asignatura
- Laboratorio de C

Criterios de valoración

Cada ejercicio tiene asociada la puntuación porcentual sobre el total de la actividad. Se valorará tanto la corrección de las respuestas como que sean completas.

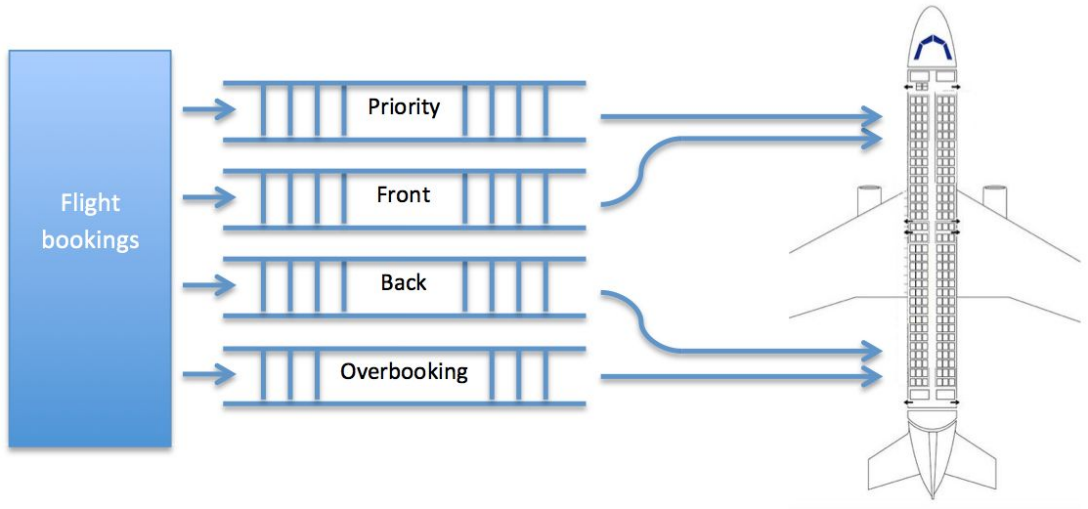
- Los ejercicios en lenguaje C tienen que compilar para ser evaluados. En tal caso, se valorará:
  - Que funcionen correctamente
  - Que se respeten los criterios de estilo (Ver la *Guía de estilo de programación en C* que tenéis a la Wiki)
  - Que el código esté comentado (preferiblemente en inglés)
  - Que las estructuras utilizadas sean las adecuadas

Descripción del proyecto

En esta práctica gestionaremos el embarque de pasajeros en el avión que los tiene que llevar a su destino. Dispondremos de una lista de pasajeros que han hecho la reserva en el avión y de varias colas de embarque. En el caso de UOCAirways, se organizan las colas de embarque de este modo:

- Los usuarios que han comprado un asiento preferente tienen asignado un asiento de las primeras filas del avión. Serán los que tengan un asiento en el primer 25% de filas del avión. Se les dará preferencia a la hora de entrar al avión, por la parte delantera.
- El resto de pasajeros que no tienen asiento preferente embarcarán por la puerta delantera o trasera del avión, atendiendo a su número de asiento. Las filas que ocupan el siguiente 25% del avión embarcan por delante también, mientras que las filas que ocupan el último 50% del avión embarcan por la puerta de atrás. Tras los pasajeros preferentes, se da paso simultáneo a estos dos conjuntos de pasajeros porque ocupan posiciones diferentes del avión y no se producirá ningún colapso por el hecho de embarcar a la vez.
- Por último, a un pasajero que no tiene asiento asignado para embarcar se le da una tarjeta de embarque provisional sin fila ni asiento asignados. De esta forma, se puede dirigir a la puerta de embarque y esperar que entre todo el mundo. Si, una vez se cierra el embarque, hay todavía asientos libres, los pasajeros de *overbooking* pueden entrar al avión, por la parte posterior, en el orden en que han recogido su tarjeta provisional.

Esquemáticamente:



Por ejemplo, con un avión de 40 filas:

- Filas 1 a 10 (25%). Asientos preferentes. Embarcan por la puerta de delante.
- Filas 11 a 20 (25%). Asientos normales. Embarcan por la puerta de delante.
- Filas 21 a 40 (50%). Asientos normales. Embarcan por la puerta de atrás.

Observad que:

- Los pasajeros entran en la lista de reservas ("*Flight bookings*") en el momento que compran el vuelo en la web de la compañía.
- Cuando se abre el periodo *de check-in*, los pasajeros obtienen su tarjeta de embarque, momento en que obtienen el número de asiento dentro del avión (si todavía hay asientos disponibles).
- Con su tarjeta de embarque y, en función del nº de fila asignada, se tienen que dirigir a la cola de embarque correspondiente, desde la que accederán al avión cuando les toque el turno y siempre en el orden de llegada a la cola.

Junto con el enunciado se os facilita un nuevo workspace Codelite basado en la solución de la práctica 1, pero con algunos cambios y evoluciones. Este workspace será la base para esta segunda práctica.

Veréis que han aparecido nuevos ficheros: **flight.c y .h** (que recoge las operaciones sobre los vuelos), **list.c y .h** (que recoge las operaciones sobre el TAD lista) y, por último, **queue.c y .h** (que recoge las operaciones sobre el TAD cola). Además, se han introducido los nuevos tipos de datos relacionados en **data.h**:

- **tBooking**, que representa una reserva de un pasajero en un vuelo, que puede ser en calidad de embarque prioritario o no.
- **tBookingList**, para representar la lista de reservas de un vuelo.
- **tBoarding**, que representa una tarjeta de embarque por la cual un pasajero tiene asignada una fila y un asiento individual del avión.
- **tBoardingQueue**, que representa una cola de embarque. Esta cola estará compuesta por varios elementos tBoarding (pasajeros con tarjeta de embarque).
- **tFlight**, que representa un vuelo de la aerolínea. Cada vuelo dispondrá de un identificador de vuelo, un avión asociado, una puerta de embarque, un origen, un destino, una fecha y una hora. Además, tendrá también la lista de reservas hechas por los pasajeros y las cuatro colas descritas anteriormente en el enunciado.
- **tFlightTable**, que contiene varios elementos de tipos tFlight. Es la relación de vuelos que la aerolínea está gestionando en un momento dado.
- El tipo **tPlane** incorpora también una tabla bidimensional que representa la distribución y ocupación de asientos dentro del avión (campos *layout* y *occupiedSeats*).
- También se incorporan otros tipos auxiliares de los anteriores como son: **tFlightId** (identificador de vuelo), **tTime** (hora del vuelo), **tAirport** (código del aeropuerto origen y destino), **tFlightStatus** (estado de un vuelo).

Además del código fuente añadido y/o modificado, veréis que aparece también un nuevo fichero de datos: el fichero de vuelos (flights.txt). Este fichero contiene la persistencia de los objetos de tipos *tFlight*, ordenados de menor a mayor fecha.

Tests

El proyecto CodeLite se presenta, como la Práctica 1, con dos modos de ejecución. No obstante, el modo *Menú* no incorpora ninguna entrada nueva que os permita probar interactivamente la gestión de los vuelos. Esto es así porque se pretende que ejecutéis y probéis vuestro programa en modo *Test*.

En modo *Test* podréis ir verificando el funcionamiento de vuestro código a medida que completéis los ejercicios. Tened en cuenta que el resultado puede ser correcto aunque el código no esté bien (es decir, el resultado no os da una corrección definitiva) y que el total de tests superados no equivale a la nota de la práctica puesto que el número de tests por apartado no se corresponde con su peso en la calificación.

Enunciado

[25%] Ejercicio 1: Comprobación de datos

El procesamiento de los vuelos se inicia en la acción *processAllFlights* (api.c). Desde esta acción se van iterando los diferentes vuelos y se van procesando uno a uno, llamando a la acción *processFlight* (api.c).

Si veis, en processFlight, las diferentes acciones de que consta el procesamiento de un vuelo, comprobareis que el primer paso es comprobar que los datos del vuelo, pasajeros y aviones son correctos. Esto se hace en la acción **checkData**, que es la que se pide implementar en este ejercicio.

En concreto, se pide que hagáis las siguientes comprobaciones en este orden:

- Comprobar que el avión del vuelo existe en la tabla de aviones y que, en caso de que exista, su número de asientos sea un múltiplo de MAX_SEATS_PER_ROW. En caso de que el avión no exista se devolverá un error ERR_NO_PLANE como valor de retorno en *retVal*. En caso de que no sea múltiple, el error devuelto será ERR_INVALID_DATA.
- Comprobar que todos los pasajeros de la lista de reservas del vuelo (*bookings*) son pasajeros registrados en la tabla de pasajeros que se recibe por parámetro. En el caso de que esto no sea así, se procederá a devolver un valor ERR_NO_PASSENGER en *retVal*.
- Comprobar que el vuelo no está iniciado. El vuelo se considerará iniciado si se da alguna de estas condiciones: tiene un estado diferente de BOOKING, tiene una puerta de embarque ya asignada o si alguna de las colas de embarque tiene ya algún pasajero. El error que se devolverá en caso de que se detecte un vuelo iniciado será ERR_STARTED_FLIGHT.

En caso de que no se detecte ninguna situación errónea de las anteriores, se devolverá en *retVal* el valor OK.

NOTAS:

- Para saber si un avión existe lo podéis buscar a partir de su identificador con la ayuda de la función *planesTable_find*.
- Tenéis disponibles operaciones para trabajar con listas de reservas en list.c.
- Para saber si un pasajero existe lo podéis buscar a partir de su identificador con la ayuda de la función *passengerTable_find*.

[25%] Ejercicio 2: Check-in del vuelo

Continuamos analizando el procesamiento de un vuelo individual (acción *processFlight*). Después de la comprobación de datos llevada a cabo en el ejercicio anterior, si todo es correcto, se localiza el avión que tendrá que hacer el trayecto y se asigna una puerta de embarque para el vuelo.

Una vez hecho esto, se inicia el check-in propiamente dicho, mediante una llamada a la acción **checkInFlight** (api.c), que es la acción que se pide implementar en este ejercicio.

En concreto, se pide:

- Cambiar el estado del vuelo a CHECKIN.
- Recorrer la lista de reservas (*bookings*), mediante las operaciones que ofrece el TAD lista para hacerlo.
- Para cada reserva (*tBooking*), hay que obtener su correspondiente tarjeta de embarque (*tBoarding*). Para hacerlo, hay que invocar la acción *assignSeat* (que ya se da implementada y que sólo hay que llamar).
- Con la tarjeta de embarque obtenida, ya sólo hay que dirigir el pasajero hacia la cola de embarque que corresponda. Habrá que comprobar el asiento asignado al pasajero y encolarlo en la cola que le corresponda (*priority*, *front*, *back* u *overbooking*) según la lógica detallada en el enunciado general.

NOTAS:

- Para poder implementar la lógica de decisión que envía un pasajero a su correspondiente cola, habrá que conocer los umbrales que permiten saber hasta qué fila llega la zona preferente, la que delimita la zona *front* y la que delimita la zona *back*. Tenéis a vuestra disposición la acción *getThresholdRows* para obtener esta información.
- La acción *assignSeat* requiere llevar la cuenta de los diferentes asientos de *priority*, *front* y *back* que se van asignando (puesto que esta acción intenta ir asignando asientos espaciadamente en el avión para repartir el peso en la medida de lo posible). Por lo tanto, hará falta que declareis e inicialiceis enteros para llevar esta cuenta, y pasarlos convenientemente en la invocación a *assignSeat*.

[25%] Ejercicio 3: Embarque de pasajeros

Siguiendo el hilo de análisis del procesamiento de un vuelo individual (acción *processFlight*), vemos que después del *checkin* se realiza una preparación del avión que tiene que empezar a admitir pasajeros (*setUpPlane*).

Una vez el avión está listo, se inicia el embarque mediante la llamada a **boardFlight** (api.c), que es la acción que se pide implementar en este ejercicio.

En concreto, se pide:

- Cambiar el estado del vuelo a BOARDING.
- Ir desencolando la cola de prioridad y acomodando los pasajeros en el avión. Para hacerlo, contáis con la acción *accomodatePassenger*.
- A continuación, ir desencolando simultáneamente las colas de pasajeros que tienen que embarcar por la parte delantera y posterior del avión. Esto quiere decir que se tendrá que dar paso alternativo a los pasajeros de una y otra cola (invocando *accomodatePassenger*).
- Como último paso, se tendrá que desencolar la cola *de overbooking* e intentar acomodar los pasajeros en esta situación dentro del avión (invocando por cada uno de ellos a la acción *accomodatePassenger*).
- Siempre que se consiga acomodar un pasajero en el avión, se le acumularán 100 puntos al pasajero en su tarjeta de fidelización (sólo en caso de que disponga de una).

NOTAS:

- La acción *accomodatePassenger* devuelve OK si se ha podido acomodar el pasajero en el avión y un valor ERR_NO_FREE_GAP en caso de que no se pueda (cosa que no tendría que producirse salvo que haya un caso de overbooking). Si un pasajero no se puede acomodar en el avión, quedará en la misma cola donde se encontraba esperando.
- Para localizar un pasajero en la tabla de pasajeros a partir de su identificador, podéis usar la función *passengerTable_find* (passenger.h). Esta función os puede resultar útil en el momento que tengáis que actualizar los puntos de fidelización del pasajero.
- Tenéis disponibles operaciones para trabajar con colas de reservas en queue.c.

[25%] Ejercicio 4: Tratamiento del overbooking

Volvemos a la acción que procesa todos los vuelos de la compañía (*processAllFlights*).

El procesamiento de un vuelo individual finaliza cuando el embarque de los pasajeros ha acabado y el vuelo queda en estado CLOSED. Pero, a pesar de que un vuelo quede cerrado, esto no quiere decir que todos sus pasajeros hayan podido embarcar. Puede haber algunos que no hayan podido obtener asiento y que haga falta reubicar en próximos vuelos.

Se pide:

a) **[10%]** Hacer una búsqueda del próximo vuelo que tenga que hacer el mismo trayecto para poder reubicar las personas que se han quedado sin volar. Para ello, debéis implementar la función **searchNextFlight** (api.c). Esta función tiene que buscar los vuelos que todavía no se han procesado y encontrar uno que haga el mismo trayecto (es decir, con los mismos aeropuertos de origen y destino).

b) **[15%]** En caso de que se encuentre un vuelo con estas características, se procederá a reubicar los pasajeros. Para hacerlo, hará falta que implementéis la acción **reallocatePassengers** (api.c), que tendrá que mover todos los pasajeros que hayan quedado en la cola de *overbooking* a la lista de reservas del vuelo encontrado en el apartado anterior. Estos pasajeros se insertarán al principio de la lista de reservas (*bookings*) en el mismo orden que tenían en la cola, para asegurar que tendrán un lugar en el próximo vuelo. Por lo tanto, pasarán por delante de los pasajeros que ya estuvieran en la lista de reservas del nuevo vuelo. Si son pasajeros poseedores de la tarjeta de fidelización de la compañía, serán insertados como reserva con prioridad. Si no lo son, serán insertados como reserva sin prioridad.

   Además, los pasajeros afectados que posean la tarjeta de fidelización de la compañía, recibirán una compensación en forma de 200 puntos en su tarjeta en el momento en que sean insertados en la lista de reservas del próximo vuelo.

NOTA:

- Para localizar un pasajero en la tabla de pasajeros a partir de su identificador, podéis usar la función *passengerTable_find* (passenger.h). Esta función os puede resultar útil en el momento que tengáis que actualizar los puntos de fidelización del pasajero.