El objetivo de esta práctica es realizar un programa en C para simular (en parte) el comportamiento de los usuarios del juego online AmongETSE, juego que se ha vuelto muy popular durante el confinamiento, y cuyo objetivo es sobrevivir en una nave como tripulantes o ser el impostor que acabará con la vida de los demás. Utilizaremos únicamente el mapa Skeld que se muestra a continuación, aunque sólo usaremos algunas habitaciones y realizaremos las tareas simples.

Realizaremos la primera versión de la práctica, en la que únicamente se creará el esqueleto de la aplicación. Se leerán los usuarios de un archivo en disco (usuarios de tu grupo de prácticas) pero se permitirá hacer altas y bajas de jugadores. Además, se seleccionarán aleatoriamente entre 4 y 10 jugadores y se les asignarán aleatoriamente el rol (Impostor o Tripulante), la tarea y la habitación en la que se realiza dicha tarea. En esta versión sólo se almacenará la última tarea asignada.

Realizaremos en la primera versión una aproximación a la base de datos de usuarios del juego, y crearemos el esqueleto de futuros desarrollos. Esta base de datos la guardaremos en el TAD árbol binario de búsqueda.

Simularemos el funcionamiento del juego a partir de un ORGANIZADOR, que es el usuario de la aplicación, y que será el único que tendrá la habilidad ADMIN (después de cada tarea ejecutada, podrá ver los jugadores que se encuentran en cada sala).

Cuando el organizador da de alta un jugador en la plataforma, debe proporcionar el campo **nombreJugador** (sólo se permite una palabra o nickname, que comenzará obligatoriamente por el carácter @). Los demás campos se asignarán cuando se seleccione la opción de "Generar datos iniciales de la partida", y son: **rol** (un carácter: I (impostor) o T (tripulante)), **descripcionTarea** (será la descripción de la última tarea asignada, una cadena de caracteres que podrá tener varias palabras), **lugarTarea** (de la lista de habitaciones posibles del mapa "Skeld", será la última ubicación en la que ha estado el jugador, será una cadena de caracteres). LA CLAVE DE ORDENACIÓN DEL ÁRBOL SERÁ EL CAMPO **nombreJugador**.

Antes de iniciarse el menú que se ejecute continuamente, la aplicación intentará leer el archivo de posibles jugadores del disco (**ETSErsG1.txt** o **ETSErsG2.txt** o **ETSErsG3.txt** o **ETSErsG4.txt**). Exista o no dicho archivo, el programa debe continuar con un menú que se ejecute continuamente hasta que el organizador elija la opción de salir. El menú contará con las siguientes opciones (letras):

a. **Alta de un jugador**

En esta función se pedirá el nickname del jugador, que debe empezar por @. Si el jugador ya está registrado, se imprimirá un mensaje indicando que ese nickname ya está siendo utilizado. Si el jugador no está registrado, se inicializarán los demás campos con la cadena " " o el carácter ' ' o con los terminadores nulos, cualquier opción que indique una inicialización de estos campos, y se introducirá en la base de datos (árbol) de jugadores (va a ser útil escribir una función privada **`_inicializarJugador()`** que realice estas operaciones).

b. **Baja de jugador**

En esta función se pide por teclado el nickname del jugador. Si no existe, se muestra un mensaje por pantalla. Si existe, se deberá eliminar dicho jugador del árbol.

l. **Listado por orden alfabética de jugadores**

Deberá hacer el recorrido en inorden del árbol mostrando el nombre de cada usuario.

g. **Generar datos iniciales de partida**

En esta función se pedirá al usuario el número de jugadores (entre 4 y 10), y el usuario podrá seleccionarlos del conjunto de jugadores dados de alta en la aplicación o bien se seleccionarán aleatoriamente del conjunto de jugadores posibles. Los jugadores que vayan a participar en el juego se guardarán en un árbol denominado Jugadores. Se calculará automáticamente el número de jugadores con el rol '**I**' (impostor) (**`=round(njugadores/5.0)`**)[i] y se generarán aleatoriamente[ii] las tareas y habitaciones a partir de una lista cerrada de **descripcionTarea** y **lugarTarea**. Como sólo se guardan los datos de la última tarea realizada, al generar nuevos datos se eliminarán los anteriores, para lo que puede ser útil una función privada **`_limpiarDatos()`**. Estas modificaciones se deben reflejar en el árbol Jugadores y en el árbol global.

> *Consejo: comenzar generando únicamente un impostor y que jueguen todos los usuarios de la aplicación (podéis dar de baja algunos si son muchos). Como la asignación del rol es aleatoria, puede que no haya ningún impostor, debéis comprobarlo y en ese caso volver a generar los roles hasta que exista al menos un impostor en la partida.*

u. **Consulta por usuario de la última tarea asignada**

En esta función se pide por teclado un nickname y se muestra la información de este: el nickname, y la última tarea realizada, si es que existe. Si el usuario no existe, se deberá mostrar un mensaje por pantalla. Puede ser útil escribir una función privada **`_imprimirJugador()`** ya que la utilizaréis en muchas de las funciones, que imprima sólo los datos de un usuario.

h. **Consulta por habitación**

En esta función se elegirá mediante un menú una de las habitaciones y se imprimirán todos los datos de los jugadores cuya última tarea se haya desarrollado en esa habitación.

s. **Salir del programa**

Se informa de la finalización del programa y se destruye el árbol antes de salir.

CONSIDERACIONES:

1. PASO 1: Se recomienda crear un proyecto en un entorno de programación (Netbeans, CLion, CodeBlocks, etc.) indicando que queréis crear el archivo main.c.

2. PASO 2: Se proporcionan en **Sesion3_iniciales.zip** el archivo abb.h de árboles binarios de búsqueda para números enteros, en el que debéis especificar el **tipoclave** (la clave de búsqueda) y el **tipoelem** (el tipo de datos en los nodos del árbol) que representen este problema. Además, debéis modificar adecuadamente las funciones en el archivo abb.c, ya que si tenéis que comparar cadenas de caracteres, en lugar de los operadores ==, > e < hay que utilizar la función **`strcmp(cad1,cad2)`** que devuelve 0 si cad1 e cad2 son iguales, un número negativo si cad1<cad2 y un número positivo si cad1>cad2 en su ordenación alfabética. Para usar esta función es necesario añadir la biblioteca **`string.h`**.

3. El programa principal debe tener un menú de opciones CON LETRAS que se repita continuamente hasta que el usuario escoja la opción de **"s. Salir"**. Este menú debe hacerse con un **switch**. El texto del menú puede estar en el main() o en una función aparte.

4. Cada opción del menú **ÚNICAMENTE** debe llamar a una función que reciba como único parámetro el árbol por valor o por referencia y realice lo que se pide.

5. Las funciones a utilizar NO deben estar en main.c, sino en el archivo **FuncionesAmongETSE.c** con la interfaz **FuncionesAmongETSE.h**. Además de contener las funciones de cada opción del menú puede incluir utilidades, como el recorrido inorden del árbol para comprobar que todas las operaciones se realizan correctamente. Fijaos bien en que funciones serán públicas y deben ir en el archivo .h y que funciones serán privadas y NO deben ir en el archivo .h.

6. Es importante, y se tendrá en cuenta en la evaluación de la práctica, que todo lo que se imprima por pantalla esté correctamente formateado de tal manera que su lectura sea sencilla. Por ejemplo, si se va a mostrar una lista de ítems se debe indicar donde comienza cada uno, bien sea con algún símbolo, con tabulaciones, etc.

Lista de tareas y habitaciones en las que se pueden desarrollar:

Para asignar una de las 8 tareas aleatoriamente, lo mejor es usar un `switch(_aleatorio(1,8))` y en cada caso copiar el texto de la tarea en el campo `descripcionTarea` de ese jugador mediante la función

`strncpy(destino,origen,longitud)`. Lo mismo con las habitaciones: si esa tarea sólo se puede realizar en una única habitación, se asigna la misma directamente al campo `lugarTarea` de ese jugador. Si hay varias posibilidades, como por ejemplo en la tarea "`Descargar datos/subir datos`", se genera un nuevo `switch(_aleatorio(1,5))` y en cada caso de este switch se copia el texto de la habitación en el campo `lugarTarea` del jugador:

Tarea 1: "Alinear la salida del motor" se desarrolla en habitación "Motores"
Tarea 2: "Calibrar distribuidor" se desarrolla en habitación "Electricidad"
Tarea 3: "Descargar datos/subir datos" se desarrolla en:
Habitación 1: "Cafetería"
Habitación 2: "Comunicaciones"
Habitación 3: "Armería"
Habitación 4: "Electricidad"
Habitación 5: "Navegación"
Tarea 4: "Desviar energía" se desarrolla en:
Habitación 1: "Navegación"
Habitación 2: "O2"
Habitación 3: "Seguridad"
Habitación 4: "Armería"
Habitación 5: "Comunicaciones"
Habitación 6: "Escudos"
Habitación 7: "Motores"
Tarea 5: "Encender escudos" se desarrolla en habitación: "Escudos"
Tarea 6: "Estabilizar dirección" se desarrolla en habitación: "Navegación"
Tarea 7: "Limpiar el filtro O2" se desarrolla en habitación: "O2"
Tarea 8: "Mapa de navegación" se desarrolla en habitación: "Navegación"

ENTREGA DE LA PRÁCTICA:

La entrega se realizará en la actividad correspondiente a cada grupo. La fecha límite, por lo general, es el día de la siguiente sesión de prácticas de vuestro grupo a la hora de inicio de la sesión, pero debéis consultar cada actividad para aseguraros.

Debéis adjuntar en un archivo comprimido denominado ApelidosNome_3.zip todos los archivos .c e .h utilizados en vuestro proyecto: main.c, FuncionesAmongETSE.h, FuncionesAmongETSE.c, abb.h y abb.c.

No utilicéis tildes ni la letra ñ en el nombre del entregable.

---

[i] Para utilizar la función **`round()`** es necesario incluir la librería matemática **`<math.h>`**

[ii] Para generar números aleatorios podéis usar la función **`srand(time(0))`** para generar una semilla aleatoria en main() y escribir una función privada **`unsigned ing _aleatorio(inf,sup)`** que devuelva un número entero aleatorio en el intervalo [inf,sup], para lo que podéis usar la función **`rand()`** que genera un número aleatorio entre 0 y 1.

Para utilizar la función **`time()`** debéis hacer un include de **`<time.h>`**.