UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Programación de sistemas y concurrencia

ANDALUCÍA TECH
Campus de Excelencia Internacional
Campus of International Excellence

Computadores A, Informática D, Software D, Matemáticas + Informática D

En este ejercicio, se va a implementar un sistema de seguridad frente a robos. El sistema va a tener registradas todas las comisarias de una ciudad, y en caso de incidente, permitirá de forma rápida localizar la más cercana. Para ello, se va a implementar una un árbol de búsqueda binario (BST) cuyos nodos tienen la siguiente estructura:

```
typedef struct Node * Tree;
typedef struct Node {
	char* name;
	double lat, lon;
	Tree left, right;
} Node;
```

El BST se va a ordenar por `name`. Los campos `lat` y `lon` son números decimales que contienen las coordenadas de la comisaria referenciada por `name`.

Ojo, `name` **no es** un array de *char*, es un puntero a un *char*.

Para desarrollar este ejercicio, se proporciona un archivo de cabecera *Tree.h* y un archivo de código fuente *main.c* con *asserts* para probar el correcto funcionamiento. Se deben implementar las siguientes funciones en el archivo *Tree.c* (se sugiere una implementación recursiva):

```
// Inicializa un árbol a vacío.
// 0.25 pts.
void inicializarArbol(Tree* ptrTree);

// Asumiendo que el árbol está ordenado (Binary Search Tree),
// se inserta un nuevo nodo ordenado por nombre con los datos
// pasados como parámetros
// 1.75 pts.
void insertarComisaria(Tree* ptrTree, char* name, double lat, double lon);

// Muestra el árbol en orden, es decir, recorrido infijo.
// 1.0 pt.
void mostrarArbol(Tree t);

// Libera toda la memoria y deja el árbol vacío.
// 1.25 pts.
void destruirArbol(Tree* ptrTree);
```

Además, el sistema debe permitir localizar la comisaria más cercana dada la latitud y longitud de un incidente, para ello vamos a usar la distancia de Manhattan:

```
// Devuelve el nombre de la comisaría más cercana dada una latitud y longitud.
// Si el árbol está vacío, se devuelve NULL.
// 2.0 pt.
char* localizarComisariaCercana(Tree t, double lat, double lon);
```

Para devolver la comisaría más cercana, debes calcular la distancia entre el incidente (lat, lon) y cada comisaría en el árbol; Si la comisaría más cercana A está en (lat', lon'), la distancia de manhattan se obtiene como:

$$dist = |lat - lat'| + |lon - lon'|$$

Encontrarás la función **fabs** en el fichero cabecera **<math.h>** para obtener el valor absoluto de un double.

Para terminar, tenemos que implementar otras dos funciones para cargar el BST desde un **archivo de texto** sin ordenar y guardarlo en un archivo binario ordenado:

```
// Carga el fichero de texto que tiene la siguiente estructura, un nombre de comisaría
nunca va a tener más de 255 caracteres de longitud:
// nombre comisaria 1; latitude1; longitude1;
// nombre comisaria 2; latitude2; longitude2;
// …
// y crea un árbol con un nodo por cada línea en ptrTree.
//
// 1.75 pts.
void cargarComisarias(FILE* filename, Tree* ptrTree);

// Guarda el arbol ordenado (recorrido infijo) en un fichero binario.
// Cada nodo será almacenado en el fichero con la siguiente estructura:
// - Un entero con la longitude del campo name.
// - Los carácteres del campo name.
// - Un double con la latitud.
// - Un double con la longitud.
//
// 2.0 pts.
void guardarBinario(char* filename, Tree tree);
```

Para implementar **guardarBinario** deberás hacer uso de alguna función auxiliar.

MUY IMPORTANTE, SI NO COMPILA, TIENES UN 0.

(Se recomienda ir haciendo copias de seguridad conforme vayan siendo implementadas las funciones y revisar todo bien antes de subir al campus).

ANEXO

Los prototipos de las funciones para manipular strings (incluidas en <string.h>) son:

char* strcpy(char *s1, char *s2): Copia los caracteres de la cadena s2 (hasta el carácter '\0', incluido) en la cadena s1. El valor devuelto es la cadena s1.

int strcmp(char *s1, char *s2): Devuelve 0 si las dos cadenas son iguales, <0 si s1 es menor que s2, y >0 si s1 es mayor que s2.

size_t strlen(const char *s): Calcula el número de caracteres de la cadena apuntada por s *¡Esta función no cuenta el carácter '\0' que finaliza la cadena!*

Los prototipos de las funciones para **manipulación de ficheros binarios** (incluidos en <stdio.h>) son los siguientes (se dan por conocidos los prototipos de las funciones de <stdlib.h> que necesites, como free o malloc):

FILE *fopen(const char *path, const char *mode): Abre el fichero especificado en el modo indicado ("rb"/"wb" para lectura/escritura binaria y "rt"/"wt" para lectura/escritura de texto). Devuelve un puntero al manejador del fichero en caso de éxito y NULL en caso de error.

int fclose(FILE *fp): Guarda el contenido del buffer y cierra el fichero especificado. Devuelve 0 en caso de éxito y -1 en caso de error.

LECTURA/ ESCRITURA TEXTO

int fscanf(FILE *stream, const char *format, ...): Lee del fichero *stream* los datos con el formato

especificado en el parámetro *format*, el resto de parámetros son las variables en las que se almacenan los

datos leídos en el formato correspondiente. La función devuelve el número de variables que se han leído

con éxito.

int fprintf(FILE *stream, const char *format, …): Escribe en el fichero *stream* los datos con el formato

especificado en el parámetro *format*. El resto de parámetros son las variables en las que se almacenan los datos que hay que escribir. La función devuelve el número de variables que se han escrito con éxito.

LECTURA/ ESCRITURA BINARIA

unsigned fread(void *ptr, unsigned size, unsigned nmemb, FILE *stream): Lee *nmemb* datos, cada uno de tamaño *size* bytes, del fichero *stream* y los almacena en la dirección apuntada por *ptr*. Devuelve el número de elementos leídos.

unsigned fwrite(const void *ptr, unsigned size, unsigned nmemb, FILE *stream): Escribe *nmemb* datos, cada uno de tamaño *size* bytes, en el fichero *stream*. Los datos se obtienen desde la dirección apuntada por *ptr*. Devuelve el número de elementos escritos.