UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

# Systems Programming and Concurrency

ANDALUCÍA TECH
Campus de Excelencia Internacional
Campus of International Excellence

# Exam of C, February 2022

## Description of the system

In this exercise you have to develop a Binary Search Tree (BST) whose nodes have the next structure:

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

The BST is sorted by the field `name`. The fields `lat` and `lon` are decimal numbers that contains the coordinates of the city referenced by `name`.
Note that `name` is not an array of char but a pointer to char.

To develop this exercise, you are given a header file **Tree.h** and source code file **driver.c** to test your. You have to implement the next functions in the file **Tree.c** (a recursive implementation is suggested):

```
// Creates an empty tree.
// 0.25 pts.
void createTree(Tree* ptrTree);

// Assuming that the tree is ordered (Binary Search Tree),
// it inserts a new Node with the data passed as parameters
// and ordered by name.
// 1.75 pts.
void insertTree(Tree* ptrTree, char* name, double lat, double lon);

// Displays the tree in order, i.e., traversing it in infix mode.
// 1.0 pt.
void showTree(Tree t);

// Frees the memory of a tree and set it to NULL.
// 1.25 pts.
void destroyTree(Tree* ptrTree);
```

In addition, this system will be used to calculate in a radar system so, in case of a hostile rocket is detected in a given coordinate, we want to know the nearest city to it. To do this, the next function must be implemented:

```
// Return the name of the nearest village to the given lat and lon coordinates.
// If the tree is empty then NULL is returned.
// The whole tree must be visited.
// 2.0 pt.
char* locateNearestCity(Tree t, double lat, double lon);
```

To return the nearest village, you have to calculate the distance between the rocket (lat, lon) and each city in the BST; if the city A is at (lat', lon'), the distance is obtained by the formula:

$$dist = \sqrt{(lat - lat')^2 + (lon - lon')^2}$$

You will find the functions **sqrt** and **pow** in the header file **<math.h>**.

To finish, we have to implement two other functions to load the BST from an unsorted text file, and to save it in a sorted binary file:

```
// Loads a text file with lines whose structure is:
// name latitude longitude
// and creates a node in a BST for each of these lines.
//
// 1.75 pts.
void loadTextFile(char* filename, Tree* ptrTree);

// Saves the tree in a sorted way in a binary file.
// Each node must be saved with the next binary structure:
// - An integer with the length of the field name
// - The characters of the field name.
// - A double with latitude.
// - A double with longitude.
//
// 2.0 pts.
void saveBinaryFile(char* filename, Tree tree);
```

To implement **saveBinaryFile** you will need the support of some auxiliary function.

**ADDENDUM**.

The headers of the functions to manage strings (<**string.h**>) are:

**char *strcpy(char *s1, const char *s2)**: copies the string pointed by s2 (including the final NULL character) into the string pointed by s1.

**int strcmp(const char *s1, const char *s2)**: compares the string pointed by s1 against the string pointed by s2. And returns:

- Greater than zero when s1>s2 (lexicographical order)
- Zero if s1==s2 (lexicographical order)
- Lower than zero when s1<s2 (lexicographical order)

**size_t strlen(const char *s)**: returns the number of characters of the string pointed to by s. The final character '\0' of the string is not counted.

Following are the headers of the functions to read and write to/from files provided by the library <**stdio.h**> (you should know by heart the headers of the required functions in <**stdlib.h**>, as **free** or **malloc**):

**FILE *fopen(const char *path, const char *mode)**: opens the filename pointed to by path using the given mode ("rb"/"wb" to binary read/write, and "rt"/"wt" to text read/write). This function returns a FILE pointer; otherwise, NULL is returned.

**int fclose(FILE *fp)**: saves the buffer content and closes the file. Returns 0 if success; otherwise -1 is returned.

**unsigned fread(void *ptr, unsigned size, unsigned nmemb, FILE *stream)**: reads nmemb data blocks, each of them composed of size bytes, from the file stream, and stores them in the address given by ptr. Returns the number of blocks read.

**unsigned fwrite(const void *ptr, unsigned size, unsigned nmemb, FILE *stream)**: writes nmemb data blocks, each of them composed of size bytes, to the file stream, taking them from the address pointed by ptr. Returns the number of blocks written.