

Meteorologica

Generado por Doxygen 1.8.13



# Índice general

<b>1</b>	<b>Índice de clases</b>	<b>1</b>
1.1	Lista de clases . . . . .	1
<b>2</b>	<b>Indice de archivos</b>	<b>3</b>
2.1	Lista de archivos . . . . .	3
<b>3</b>	<b>Documentación de las clases</b>	<b>5</b>
3.1	Referencia de la Estructura City . . . . .	5
3.1.1	Descripción detallada . . . . .	5
3.1.2	Documentación de los datos miembro . . . . .	5
3.1.2.1	city . . . . .	5
3.1.2.2	memory_reserved . . . . .	5
3.1.2.3	meteoData_array . . . . .	6
3.1.2.4	n_meteoData_entries . . . . .	6
3.2	Referencia de la Estructura Meteo . . . . .	6
3.2.1	Descripción detallada . . . . .	6
3.2.2	Documentación de los datos miembro . . . . .	6
3.2.2.1	city . . . . .	6
3.2.2.2	cloudiness . . . . .	7
3.2.2.3	date . . . . .	7
3.2.2.4	max_temp . . . . .	7
3.2.2.5	min_temp . . . . .	7
3.2.2.6	precipitations . . . . .	7

<b>4 Documentación de archivos</b>	<b>9</b>
4.1 Referencia del Archivo helpers.c	9
4.1.1 Descripción detallada	10
4.1.2 Documentación de las funciones	10
4.1.2.1 celsiusToFahrenheit()	10
4.1.2.2 compareDates()	10
4.1.2.3 compareString()	11
4.1.2.4 fixDecimalNotation()	11
4.1.2.5 getIntFromUserInput()	11
4.1.2.6 getStringFromUserInput()	12
4.1.2.7 replace()	12
4.1.2.8 split()	13
4.1.2.9 storeToFile()	13
4.1.2.10 strCelsiusToStrFahrenheit()	13
4.1.2.11 stringToFloat()	14
4.1.2.12 trim()	14
4.2 Referencia del Archivo helpers.h	15
4.2.1 Descripción detallada	15
4.2.2 Documentación de las funciones	16
4.2.2.1 celsiusToFahrenheit()	16
4.2.2.2 compareDates()	16
4.2.2.3 compareString()	16
4.2.2.4 fixDecimalNotation()	17
4.2.2.5 getIntFromUserInput()	17
4.2.2.6 getStringFromUserInput()	17
4.2.2.7 replace()	18
4.2.2.8 split()	18
4.2.2.9 storeToFile()	19
4.2.2.10 strCelsiusToStrFahrenheit()	19
4.2.2.11 stringToFloat()	20

4.2.2.12	trim()	20
4.3	Referencia del Archivo interfaces.c	20
4.3.1	Descripción detallada	21
4.3.2	Documentación de las funciones	21
4.3.2.1	showForecast()	21
4.4	Referencia del Archivo interfaces.h	21
4.4.1	Descripción detallada	22
4.4.2	Documentación de las funciones	22
4.4.2.1	showForecast()	22
4.5	Referencia del Archivo main.c	22
4.5.1	Descripción detallada	23
4.6	Referencia del Archivo meteo.c	23
4.6.1	Descripción detallada	23
4.6.2	Documentación de las funciones	24
4.6.2.1	binarySearchCities()	24
4.6.2.2	binarySearchDates()	24
4.6.2.3	customMeteoDataParser()	25
4.6.2.4	loadData()	25
4.6.2.5	printMeteoEntry()	26
4.7	Referencia del Archivo meteo.h	27
4.7.1	Descripción detallada	27
4.7.2	Documentación de las funciones	27
4.7.2.1	binarySearchCities()	27
4.7.2.2	binarySearchDates()	28
4.7.2.3	customMeteoDataParser()	28
4.7.2.4	loadData()	29
4.7.2.5	printMeteoEntry()	29



# Capítulo 1

## Índice de clases

### 1.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

<a href="#">City</a>	Estructura de datos que contiene la información de la ciudad almacenada . . . . .	<a href="#">5</a>
<a href="#">Meteo</a>	Estructura de datos que contienen la información meteorológica de una ciudad en una fecha determinada . . . . .	<a href="#">6</a>





## Capítulo 2

# Indice de archivos

### 2.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

<a href="#">helpers.c</a>		
	En este fichero están definidas las funciones declaradas en <a href="#">helpers.h</a> . . . . .	9
<a href="#">helpers.h</a>		
	En este fichero están declaradas aquellas funciones cuyo objetivo es de caracter general y pueden ser invocadas en multiples situaciones . . . . .	15
<a href="#">interfaces.c</a>		
	En este fichero están definidas las funciones declaradas en <a href="#">interfaces.h</a> . . . . .	20
<a href="#">interfaces.h</a>		
	En este fichero están declaradas aquellas funciones que tienen como objetivo interactuar con el usuario de la aplicación . . . . .	21
<a href="#">main.c</a>		
	En este fichero se inicializa el programa . . . . .	22
<a href="#">meteo.c</a>		
	En este fichero están definidas las funciones declaradas en <a href="#">meteo.h</a> . . . . .	23
<a href="#">meteo.h</a>		
	En este fichero están declaradas aquellas funciones que están relacionadas con el manejo y procesamiento de los datos relacionados con la meteorología . . . . .	27



## Capítulo 3

# Documentación de las clases

### 3.1. Referencia de la Estructura City

Estructura de datos que contiene la información de la ciudad almacenada.

```
#include <meteo.h>
```

#### Atributos públicos

- char `city` [100]
- int `n_meteoData_entries`
- int `memory_reserved`
- struct `Meteo` \* `meteoData_array`

#### 3.1.1. Descripción detallada

Estructura de datos que contiene la información de la ciudad almacenada.

#### 3.1.2. Documentación de los datos miembro

##### 3.1.2.1. `city`

```
char City::city[100]
```

Nombre de la ciudad

##### 3.1.2.2. `memory_reserved`

```
int City::memory_reserved
```

Número de espacio reservado para entradas meteorológicas

### 3.1.2.3. meteoData\_array

```
struct Meteo* City::meteoData_array
```

Array con la información meteorológica para la ciudad

### 3.1.2.4. n\_meteoData\_entries

```
int City::n_meteoData_entries
```

Número de entradas meteorológicas almacenadas para esta ciudad

La documentación para esta estructura fue generada a partir del siguiente fichero:

- [meteo.h](#)

## 3.2. Referencia de la Estructura Meteo

Estructura de datos que contienen la información meteorológica de una ciudad en una fecha determinada.

```
#include <meteo.h>
```

### Atributos públicos

- char [date](#) [50]
- char [city](#) [100]
- char [max\\_temp](#) [50]
- char [min\\_temp](#) [50]
- char [precipitations](#) [50]
- char [cloudiness](#) [50]

### 3.2.1. Descripción detallada

Estructura de datos que contienen la información meteorológica de una ciudad en una fecha determinada.

### 3.2.2. Documentación de los datos miembro

#### 3.2.2.1. city

```
char Meteo::city[100]
```

Nombre de la ciudad de la entrada meteorológica

**3.2.2.2. cloudiness**

```
char Meteo::cloudiness[50]
```

Nubosidad de lea entrada meteorológica

**3.2.2.3. date**

```
char Meteo::date[50]
```

Fecha de la entrada meteorológica

**3.2.2.4. max\_temp**

```
char Meteo::max_temp[50]
```

Temperatura máxima de la entrada meteorológica

**3.2.2.5. min\_temp**

```
char Meteo::min_temp[50]
```

Temperatura mínima de la entrada meteorológica

**3.2.2.6. precipitations**

```
char Meteo::precipitations[50]
```

Precipitaciones de la entrada meteorológica

La documentación para esta estructura fue generada a partir del siguiente fichero:

- [meteo.h](#)



## Capítulo 4

# Documentación de archivos

### 4.1. Referencia del Archivo helpers.c

En este fichero están definidas las funciones declaradas en [helpers.h](#).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include "helpers.h"
```

#### Funciones

- char \*\* [split](#) (char \*string, const char delimiter)  
*La función [split\(\)](#) divide el string recibido en substring utilizando como divisor un delimitador. Ejemplo [split](#)("Esto es una prueba", ' ') obtendríamos un array con 4 strings "Esto", "es", "una", "prueba".*
- void [fixDecimalNotation](#) (char \*string)  
*La función [fixDecimalNotation\(\)](#) corrige la notación de los números decimales. Sustituye los decimales expresados con ',' por '.' Ejemplo: 12,23 -> 12.23.*
- void [trim](#) (char \*string)  
*La función [trim\(\)](#) elimina los espacios en blanco ' ' y otros caracteres especiales tanto al comienzo como al final del string recibido.*
- int [getIntFromUserInput](#) (int \*number)  
*La función [getIntFromUserInput\(\)](#) recibe una entrada por teclado y la convierte en INT cuando sea posible.*
- int [getStringFromUserInput](#) (char \*string, int size)  
*La función [getStringFromUserInput\(\)](#) recibe una entrada por teclado y la almacena.*
- int [compareString](#) (void const \*str1, void const \*str2)  
*La función [compareString\(\)](#) compara dos string. Internamente utiliza la función [strcmp\(\)](#). La razón de ser de esta función es para utilizarla como comparador en la función de ordenación [qsort\(\)](#)*
- int [compareDates](#) (char \*date1, char \*date2)  
*La función [compareDates\(\)](#) compara dos fechas en formato AAAA/MM/DD (Año/Mes/Día)*
- int [strCelsiusToStrFahrenheit](#) (char \*celsius, char \*\*fahrenheit)  
*La función [strCelsiusToStrFahrenheit\(\)](#) recibe grados celsius en formato string para transformarlo en grados Fahrenheit también en formato string.*
- int [stringToFloat](#) (double \*value, char \*data)

La función `stringToFloat()` convierte una cadena de caracteres en float si es posible.

- `double celsiusToFahrenheit` (`double celsius`)

La función `celsiusToFahrenheit` convierte los grados Celsius introducidos en Fahrenheit.

- `int storeToFile` (`char *filename`, `char *data`)

La función `storeToFile()` almacena en un fichero los datos proporcionados. Esta función creará el fichero si no existe o lo sobrescribirá en caso de que existiera.

- `void replace` (`char *string`, `char c1`, `char c2`)

La función `replace()`, reemplaza la el caracter `c1` por `c2` en toda la string.

#### 4.1.1. Descripción detallada

En este fichero están definidas las funciones declaradas en [helpers.h](#).

#### 4.1.2. Documentación de las funciones

##### 4.1.2.1. `celsiusToFahrenheit()`

```
double celsiusToFahrenheit (
    double celsius )
```

La función `celsiusToFahrenheit` convierte los grados Celsius introducidos en Fahrenheit.

##### Parámetros

<code>celsius</code>	Se trata de un double que contiene los grados celsius a convertir
----------------------	---

##### Devuelve

La función `celsiusToFahrenheit` devuelve un double con la conversión de Celsius a Fahrenheit

##### 4.1.2.2. `compareDates()`

```
int compareDates (
    char * date1,
    char * date2 )
```

La función `compareDates()` compara dos fechas en formato AAAA/MM/DD (Año/Mes/Día)

##### Parámetros

<code>date1</code>	Se trata de un string ( <code>char *</code> ) que contiene la primera fecha a comparar.
<code>date2</code>	Se trata de un string ( <code>char *</code> ) que contiene la segunda fecha a comparar.



**Devuelve**

Si `date1 > date2` devuelve un 1; Si `date1 < date2` devuelve un -1; Si `date1 == date2` devuelve un 0;

**4.1.2.3. `compareString()`**

```
int compareString (
    void const * str1,
    void const * str2 )
```

La función [compareString\(\)](#) compara dos string. Internamente utiliza la función `strcmp()`. La razón de ser de esta función es para utilizarla como comparador en la función de ordenación `qsort()`

**Parámetros**

<i>str1</i>	Se trata de un string (char *) que contiene la primera string a comparar.
<i>str2</i>	Se trata de un string (char *) que contiene la segunda string a comparar.

**Devuelve**

Si `str1 > str2` devuelve un número positivo; Si `str1 < str2` devuelve un número negativo; Si `str1 == str2` devuelve un cero;

**4.1.2.4. `fixDecimalNotation()`**

```
void fixDecimalNotation (
    char * string )
```

La función [fixDecimalNotation\(\)](#) corrige la notación de los números decimales. Sustituye los decimales expresados con ',' por '.' Ejemplo: 12,23 -> 12.23.

**Parámetros**

<i>string</i>	Se trata de un string (char *) que contiene el número al que se le quiere corregir el símbolo de decimal.
---------------	---

**4.1.2.5. `getIntFromUserInput()`**

```
int getIntFromUserInput (
    int * number )
```

La función [getIntFromUserInput\(\)](#) recibe una entrada por teclado y la convierte en INT cuando sea posible.

**Parámetros**

<i>number</i>	Se trata de un puntero a entero (int *) en el cuál se almacenará el número recibido.
---------------	--

**Devuelve**

La función devuelve un 0 en caso de que se haya podido recibir y convertir la entrada a entero. En caso de error, devolverá -1.

**4.1.2.6. getStringFromUserInput()**

```
int getStringFromUserInput (
    char * string,
    int size )
```

La función [getStringFromUserInput\(\)](#) recibe una entrada por teclado y la almacena.

**Parámetros**

<i>string</i>	Se trata de un string (char *) en el cuál se almacenará el la cadena de texto recibida.
<i>size</i>	Entero con la tamaño máximo de la cadena que se desea recibir.

**Devuelve**

La función devuelve un entero indicando la longitud del string recibido. En caso de error devolverá -1.

**4.1.2.7. replace()**

```
int replace (
    char * string,
    char c1,
    char c2 )
```

La función [replace\(\)](#), reemplaza la el caracter c1 por c2 en toda la string.

**Parámetros**

<i>string</i>	Se trata de un string que contiene los caracteres que se quieren reemplazar
<i>c1</i>	Se trata de caracter que se quiere reemplazar por otro
<i>c2</i>	Se trata de caracter por el cuál se va a reemplazar

#### 4.1.2.8. split()

```
char ** split (
    char * string,
    const char delimiter )
```

La función `split()` divide el string recibido en substring utilizando como divisor un delimitador. Ejemplo `split("Esto es una prueba", ' ')` obtendríamos un array con 4 strings "Esto", "es", "una", "prueba".

##### Parámetros

<i>string</i>	Se trata de un string (char *) que contiene la string que se quiere dividir.
<i>delimiter</i>	Se trata de un caracter (char) que contiene el delimitador por el cual se dividirá el string.

##### Devuelve

La función `split()` devuelve un array de strings (char \*\*) en el cual cada posición del array contiene una de las divisiones del string introducido.

##### Atención

Es necesario liberar la memoria reservada por la función `split()` del parametro devuelto. Para ello liberaremos la memoria reservada para cada string devuelto en el array y luego liberaremos la memoria del array.

#### 4.1.2.9. storeToFile()

```
int storeToFile (
    char * filename,
    char * data )
```

La función `storeToFile()` almacena en un fichero los datos proporcionados. Esta función creará el fichero si no existe o lo sobrescribirá en caso de que existiera.

##### Parámetros

<i>filename</i>	Se trata de un string (char *) que contiene el nombre del fichero donde se va a guardar los datos
<i>data</i>	Se trata de un string (char *) que contiene los datos que se van a almacenar en el fichero

##### Devuelve

La función `storeToFile` devuelve un entero con un valor de 0 en caso de éxito y un -1 en caso de error

#### 4.1.2.10. strCelsiusToStrFahrenheit()

```
int strCelsiusToStrFahrenheit (
    char * Celsius,
    char ** Fahrenheit )
```

La función `strCelsiusToStrFahrengait()` recibe grados celsiud en formato string para transofrmalor en grados Fahrenheit también en formato string.

#### Parámetros

<i>celsius</i>	Se trata de un string (char *) que contiene los grados celsiud a convertir
<i>fahrenheit</i>	Se trata de un puntero a string (char **) que contendrá el resultado de la conversión de Celsiud a Fahrenheit

#### Devuelve

La función `strCelsiudToStrFahrenheit()` devuelve un entero con valor 0 si la operación a resultado con éxito y un -1 si se produce un error.

#### Atención

Es necesario liberar la memoria reservada por la función `strCelsiusToStrFahrenheit()`. Para ello liberaremos la mermoria del puntero a string `fahrenheit`.

#### 4.1.2.11. `stringToFloat()`

```
int stringToFloat (
    double * value,
    char * data )
```

La función `stringToFloat()` convierte una cadena de caracteres en float si es posible.

#### Parámetros

<i>value</i>	Se trata de un double donde se almacenará el número en caso de éxito;
<i>data</i>	Se trata de un string (char *) que contiene los datos que se desean convertir

#### Devuelve

La función `stringToFloat` devuelve un entero con un 0 en caso de éxito y un -1 si hay algún error.

#### 4.1.2.12. `trim()`

```
void trim (
    char * string )
```

La función `trim()` elimina los espacios en blanco ' ' y otros caracteres especiales tanto al comienzo como al final del string recibido.

## Parámetros

<code>string</code>	Se trata de un string (char *) a la que le queremos quitar los espacios en blanco.
---------------------	--

## 4.2. Referencia del Archivo helpers.h

En este fichero están declaradas aquellas funciones cuyo objetivo es de caracter general y pueden ser invocadas en multiples situaciones.

### Funciones

- char \*\* [split](#) (char \*string, const char delimiter)  
*La función [split\(\)](#) divide el string recibido en substring utilizando como divisor un delimitador. Ejemplo [split](#)("Esto es una prueba", ' ') obtendriamos un array con 4 strings "Esto", "es", "una", "prueba".*
- void [fixDecimalNotation](#) (char \*string)  
*La función [fixDecimalNotation\(\)](#) corrige la notación de los números decimales. Sustituye los decimales expresados con ',' por '.' Ejemplo: 12,23 -> 12.23.*
- void [trim](#) (char \*string)  
*La función [trim\(\)](#) elimina los espacios en blanco ' ' y otros caracteres especiales tanto al comienzo como al final del string recibido.*
- int [getIntFromUserInput](#) (int \*number)  
*La función [getIntFromUserInput\(\)](#) recibe una entrada por teclado y la convierte en INT cuando sea posible.*
- int [getStringFromUserInput](#) (char \*string, int size)  
*La función [getStringFromUserInput\(\)](#) recibe una entrada por teclado y la almacena.*
- int [compareString](#) (void const \*str1, void const \*str2)  
*La función [compareString\(\)](#) compara dos string. Internamente utiliza la funcion [strcmp\(\)](#). La razón de ser de esta función es para utilizarla como comparador en la función de ordenación [qsort\(\)](#)*
- int [compareDates](#) (char \*date1, char \*date2)  
*La función [compareDates\(\)](#) compara dos fechas en formato AAAA/MM/DD (Año/Mes/Día)*
- int [strCelsiusToStrFahrenheit](#) (char \*celsius, char \*\*fahrenheit)  
*La función [strCelsiusToStrFahrenheit\(\)](#) recibe grados celsius en formato string para transformarlo en grados Fahrenheit también en formato string.*
- int [stringToFloat](#) (double \*value, char \*data)  
*La función [stringToFloat\(\)](#) convierte una cadena de caracteres en float si es posible.*
- double [celsiusToFahrenheit](#) (double celsius)  
*La función [celsiusToFahrenheit](#) convierte los grados Celsius introducidos en Fahrenheit.*
- int [storeToFile](#) (char \*filename, char \*data)  
*La función [storeToFile\(\)](#) almacena en un fichero los datos proporcionados. Esta función creará el fichero si no existe o lo sobrescribirá en caso de que existiera.*
- void [replace](#) (char \*string, char c1, char c2)  
*La función [replace\(\)](#), reemplaza el caracter c1 por c2 en toda la string.*

### 4.2.1. Descripción detallada

En este fichero están declaradas aquellas funciones cuyo objetivo es de caracter general y pueden ser invocadas en multiples situaciones.

## 4.2.2. Documentación de las funciones

### 4.2.2.1. celsiusToFahrenheit()

```
double celsiusToFahrenheit (
    double celsius )
```

La función `celsiusToFahrenheit` convierte los grados Celcius introducidos en Fahrenheit.

#### Parámetros

<i>celsius</i>	Se trata de un double que contiene los grados celsius a convertir
----------------	---

#### Devuelve

La función `celsiusToFahrenheit` devuelve un double con la conversión de Celsius a Fahrenheit

### 4.2.2.2. compareDates()

```
int compareDates (
    char * date1,
    char * date2 )
```

La función `compareDates()` compara dos fechas en formato AAAA/MM/DD (Año/Mes/Día)

#### Parámetros

<i>date1</i>	Se trata de un string (char *) que contiene la primera fecha a comparar.
<i>date2</i>	Se trata de un string (char *) que contiene la segunda fecha a comparar.

#### Devuelve

Si `date1 > date2` devuelve un 1; Si `date1 < date2` devuelve un -1; Si `date1 == date2` devuelve un 0;

### 4.2.2.3. compareString()

```
int compareString (
    void const * str1,
    void const * str2 )
```

La función `compareString()` compara dos string. Internamente utiliza la funcion `strcmp()`. La razón de ser de esta función es para utilizarla como comparador en la función de ordenación `qsort()`

**Parámetros**

<i>str1</i>	Se trata de un string (char *) que contiene la primera string a comparar.
<i>str2</i>	Se trata de un string (char *) que contiene la segunda string a comparar.

**Devuelve**

Si  $str1 > str2$  devuelve un número positivo; Si  $str1 < str2$  devuelve un número negativo; Si  $str1 == str2$  devuelve un cero;

**4.2.2.4. fixDecimalNotation()**

```
void fixDecimalNotation (
    char * string )
```

La función [fixDecimalNotation\(\)](#) corrige la notación de los números decimales. Sustituye los decimales expresados con ',' por '.' Ejemplo: 12,23 -> 12.23.

**Parámetros**

<i>string</i>	Se trata de un string (char *) que contiene el número al que se le quiere corregir el simbolo de decimal.
---------------	---

**4.2.2.5. getIntFromUserInput()**

```
int getIntFromUserInput (
    int * number )
```

La función [getIntFromUserInput\(\)](#) recibe una entrada por teclado y la convierte en INT cuando sea posible.

**Parámetros**

<i>number</i>	Se trata de un puntero a entero (int *) en el cuál se almacenará el número recibido.
---------------	--

**Devuelve**

La función devuelve un 0 en caso de que se haya podido recibir y convertir la entrada a entero. En caso de error, devolverá -1.

**4.2.2.6. getStringFromUserInput()**

```
int getStringFromUserInput (
    char * string,
    int size )
```

La función `getStringFromUserInput()` recibe una entrada por teclado y la almacena.

#### Parámetros

<i>string</i>	Se trata de un string (char *) en el cuál se almacenará el la cadena de texto recibida.
<i>size</i>	Entero con la tamaño máximo de la cadena que se desea recibir.

#### Devuelve

La función devuelve un entero indicando la longitud del string recibido. En caso de error devolverá -1.

#### 4.2.2.7. `replace()`

```
void replace (
    char * string,
    char c1,
    char c2 )
```

La función `replace()`, reemplaza la el caracter c1 por c2 en toda la string.

#### Parámetros

<i>string</i>	Se trata de un string que contiene los caracteres que se quieren reemplazar
<i>c1</i>	Se trata de caracter que se quiere reemplazar por otro
<i>c2</i>	Se trata de caracter por el cuál se va a reemplazar

#### 4.2.2.8. `split()`

```
char** split (
    char * string,
    const char delimiter )
```

La función `split()` divide el string recibido en substring utilizando como divisor un delimitador. Ejemplo `split("Esto es una prueba", ' ')` obtendríamos un array con 4 strings "Esto", "es", "una", "prueba".

#### Parámetros

<i>string</i>	Se trata de un string (char *) que contiene la string que se quiere dividir.
<i>delimiter</i>	Se trata de un caracter (char) que contiene el delimitador por el cual se dividirá el string.

#### Devuelve

La función `split()` devuelve un array de strings (char \*\*) en el cual cada posición del array contiene una de las divisiones del string introducido.



**Atención**

Es necesario liberar la memoria reservada por la función `split()` del parametro devuelto. Para ello liberaremos la memoria reservada para cada string devuelto en el array y luego liberaremos la memoria del array.

**4.2.2.9. storeToFile()**

```
int storeToFile (
    char * filename,
    char * data )
```

La función `storeToFile()` almacena en un fichero los datos proporcionados. Esta función creará el fichero si no existe o lo sobrescribirá en caso de que existiera.

**Parámetros**

<i>filename</i>	Se trata de un string (char *) que contiene el nombre del fichero donde se va a guardar los datos
<i>data</i>	Se trata de un string (char *) que contiene los datos que se van a almacenar en el fichero

**Devuelve**

La función `storeToFile` devuelve un entero con un valor de 0 en caso de éxito y un -1 en caso de error

**4.2.2.10. strCelsiusToStrFahrenheit()**

```
int strCelsiusToStrFahrenheit (
    char * celsius,
    char ** fahrenheit )
```

La función `strCelsiusToStrFahrenheit()` recibe grados celsius en formato string para transformarlo en grados Fahrenheit también en formato string.

**Parámetros**

<i>celsius</i>	Se trata de un string (char *) que contiene los grados celsius a convertir
<i>fahrenheit</i>	Se trata de un puntero a string (char **) que contendrá el resultado de la conversión de Celsius a Fahrenheit

**Devuelve**

La función `strCelsiusToStrFahrenheit()` devuelve un entero con valor 0 si la operación a resultado con éxito y un -1 si se produce un error.

### Atención

Es necesario liberar la memoria reservada por la función [strCelsiusToStrFahrenheit\(\)](#). Para ello liberaremos la mermoria del puntero a string fahrenheit.

#### 4.2.2.11. [stringToFloat\(\)](#)

```
int stringToFloat (
    double * value,
    char * data )
```

La función [stringToFloat\(\)](#) convierte una cadena de caracteres en float si es posible.

#### Parámetros

<i>value</i>	Se trata de un double donde se almacenará el número en caso de éxito;
<i>data</i>	Se trata de un string (char *) que contiene los datos que se desean convertir

### Devuelve

La función [stringToFloat](#) devuelve un entero con un 0 en caso de éxito y un -1 si hay algún error.

#### 4.2.2.12. [trim\(\)](#)

```
void trim (
    char * string )
```

La función [trim\(\)](#) elimina los espacios en blanco ' ' y otros caracteres especiales tanto al comienzo como al final del string recibido.

#### Parámetros

<i>string</i>	Se trata de un string (char *) a la que le queremos quitar los espacios en blanco.
---------------	--

## 4.3. Referencia del Archivo [interfaces.c](#)

En este fichero están definidas las funciones declaradas en [interfaces.h](#).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "interfaces.h"
#include "meteo.h"
#include "helpers.h"
```

## Funciones

- int **interactiveMenu** (char \*city, char \*date, char \*unit, int \*days, int \*printJSON, int \*saveJSON)
- int **showForecast** (struct **Meteo** \*forecast, int n\_data, char unit)

La función **showForecast()** imprime por pantalla la información meteorológica proporcionada.

### 4.3.1. Descripción detallada

En este fichero están definidas las funciones declaradas en [interfaces.h](#).

### 4.3.2. Documentación de las funciones

#### 4.3.2.1. showForecast()

```
int showForecast (
    struct Meteo * forecast,
    int n_data,
    char unit )
```

La función **showForecast()** imprime por pantalla la información meteorológica proporcionada.

#### Parámetros

<i>forecast</i>	Se trata de un array de estructuras de tipo <b>Meteo</b> (struct <b>Meteo</b> *) que contiene la información meteorológica.
<i>n_data</i>	Se trata de un entero que contiene la cantidad de entradas meteorológicas que contiene el array forecast.
<i>unit</i>	Se trata de una char que contiene la unidad en la que se representarán las unidades. Estas pueden ser 'C' para Celsius o 'F' para Fahrenheit.

#### Devuelve

La función showForecast devuelve un entero con valor 0 si se ha ejecutado con éxito, y un -1 si se ha producido un error.

## 4.4. Referencia del Archivo interfaces.h

En este fichero están declaradas aquellas funciones que tienen como objetivo interactuar con el usuario de la aplicación.

```
#include "meteo.h"
```

## Funciones

- int **interactiveMenu** (char \*city, char \*date, char \*unit, int \*days, int \*printJSON, int \*saveJSON)
- int **showForecast** (struct **Meteo** \*forecast, int n\_data, char unit)

La función **showForecast()** imprime por pantalla la información meteorológica proporcionada.

### 4.4.1. Descripción detallada

En este fichero están declaradas aquellas funciones que tienen como objetivo interactuar con el usuario de la aplicación.

### 4.4.2. Documentación de las funciones

#### 4.4.2.1. showForecast()

```
int showForecast (
    struct Meteo * forecast,
    int n_data,
    char unit )
```

La función **showForecast()** imprime por pantalla la información meteorológica proporcionada.

#### Parámetros

<i>forecast</i>	Se trata de un array de estructuras de tipo <b>Meteo</b> (struct <b>Meteo</b> *) que contiene la información meteorológica.
<i>n_data</i>	Se trata de un entero que contiene la cantidad de entradas meteorológicas que contiene el array forecast.
<i>unit</i>	Se trata de una char que contiene la unidad en la que se representarán las unidades. Estas pueden ser 'C' para Celsius o 'F' para Fahrenheit.

#### Devuelve

La función **showForecast** devuelve un entero con valor 0 si se ha ejecutado con éxito, y un -1 si se ha producido un error.

## 4.5. Referencia del Archivo main.c

En este fichero se inicializa el programa.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "meteo.h"
#include "interfaces.h"
#include "helpers.h"
```

## Funciones

- int **main** ()

### 4.5.1. Descripción detallada

En este fichero se inicializa el programa.

## 4.6. Referencia del Archivo meteo.c

En este fichero están definidas las funciones declaradas en [meteo.h](#).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "meteo.h"
#include "helpers.h"
```

## defines

- #define **CITIES\_BUFFER\_INCREMENT** 5
- #define **METEO\_DATA\_BUFFER\_INCREMENT** 10

## Funciones

- void [printMeteoEntry](#) (struct [Meteo](#) meteoData)  
*La función [printMeteoEntry\(\)](#) muestra por pantalla la información de la entrada meteorológica recibida.*
- int [loadData](#) (const char \*filename, struct [City](#) \*\*cities)  
*La función [loadData\(\)](#) carga datos desde un fichero de texto en formato csv a un array de ciudades de tipo [City](#) (struct [City](#) \*). Para cada entrada de este array de ciudades, se encuentran los datos meteorológicos correspondientes a cada ciudad. Las ciudades son ordenadas por orden Alfabético para facilitar su búsqueda.*
- struct [Meteo](#) [customMeteoDataParser](#) (char \*line)  
*La función [customMeteoDataParser\(\)](#) recibe una string con los datos Meteorológicos en formato csv y los convierte a una estructura de datos de tipo struct [Meteo](#).*
- int [binarySearchCities](#) (char \*city, struct [City](#) \*cities, int numCities)  
*La función [binarySearchCities\(\)](#) realiza una búsqueda binaria en el array de ciudades.*
- int [binarySearchDates](#) (char \*date, struct [Meteo](#) \*meteoData, int numDataEntries)  
*La función [binarySearchCities\(\)](#) realiza una búsqueda binaria en el array de ciudades.*
- int [getForecast](#) (struct [City](#) \*cities, int numCities, struct [Meteo](#) \*\*forecast, char \*city, char \*date, int days)
- int [meteoDataToJSON](#) (struct [Meteo](#) \*meteoData, int meteoDataLength, char \*\*jsonData, char unit)

### 4.6.1. Descripción detallada

En este fichero están definidas las funciones declaradas en [meteo.h](#).

## 4.6.2. Documentación de las funciones

### 4.6.2.1. `binarySearchCities()`

```
int binarySearchCities (
    char * city,
    struct City * cities,
    int numCities )
```

La función `binarySearchCities()` realiza una búsqueda binaria en el array de ciudades.

#### Parámetros

<i>city</i>	Se trata de un string (char *) que contiene el nombre de la ciudad a buscar
<i>cities</i>	Se trata de un array de estructuras de tipo <code>City</code> (struct <code>City</code> *) que contiene las ciudades sobre las que se realizará la búsqueda
<i>numCities</i>	Se trata de un entero con el número de ciudades almacenadas

#### Devuelve

La función `binarySearchCities` devuelve la posición en el array de la ciudad si la ciudad es encontrada. Si no se ha encontrado devolverá -1.

### 4.6.2.2. `binarySearchDates()`

```
int binarySearchDates (
    char * date,
    struct Meteo * meteoData,
    int numDataEntries )
```

La función `binarySearchCities()` realiza una búsqueda binaria en el array de ciudades.

#### Parámetros

<i>city</i>	Se trata de un puntero a una string (char *) que contiene el nombre de la ciudad a buscar
<i>cities</i>	Se trata de un array de estructuras de tipo <code>City</code> (struct <code>City</code> *) que contiene las ciudades sobre las que se realizará la búsqueda
<i>numCities</i>	Se trata de un entero con el número de ciudades almacenadas

#### Devuelve

La función `binarySearchCities` devuelve la posición en el array de la ciudad si la ciudad es encontrada. Si no se ha encontrado devolverá -1.

#### 4.6.2.3. customMeteoDataParser()

```
struct Meteo customMeteoDataParser (
    char * line )
```

La función `customMeteoDataParser()` recibe una string con los datos Meteorológicos en formato csv y los convierte a una estructura de datos de tipo struct `Meteo`.

##### Parámetros

<i>line</i>	Se trata de un string (char *) que contiene la string con los datos que se quieren cargar.
-------------	--

##### Devuelve

La función `customMeteoDataParser()` devuelve una estructura de datos de tipo `Meteo` (struct `Meteo`) con los datos cargados del string line

#### 4.6.2.4. loadData()

```
int loadData (
    const char * filename,
    struct City ** cities )
```

La función `loadData()` carga datos desde un fichero de texto en formato csv a un array de ciudades de tipo `City` (struct `City *`). Para cada entrada de este array de ciudades, se encuentran los datos meteorológicos correspondientes a cada ciudad. Las ciudades son ordenadas por orden Alfabético para facilitar su búsqueda.

##### Parámetros

<i>filename</i>	Se trata de un string (char *) que contiene el nombre del fichero de texto a cargar.
<i>cities</i>	Se trata de un puntero a un array de estructuras de tipo <code>City</code> (struct <code>City **</code> ) en el que se van a almacenar los datos cargados por ciudades

##### Devuelve

La función `loadData()` devuelve el número de elementos cargados si tiene éxito. En caso de error devolverá -1.

##### Atención

Es necesario liberar la memoria reservada en la función `loadData()` del parametro cities. Para ello liberaremos la información para cada entrada meteorológica para cada ciudad y posteriormente liberaremos la memoria reservada para cada ciudad.

#### 4.6.2.5. printMeteoEntry()

```
void printMeteoEntry (
    struct Meteo meteoData )
```

La funcion `printMeteoEntry()` muestra por pantalla la información de la entrada meteorológica recibida.



## Parámetros

<code>meteoData</code>	Se trata de una estructura de datos de tipo <a href="#">Meteo</a> la cual contiene la información a mostrar por pantalla
------------------------	--

## 4.7. Referencia del Archivo meteo.h

En este fichero están declaradas aquellas funciones que están relacionadas con el manejo y procesamiento de los datos relacionados con la meteorología.

### Clases

- struct [City](#)  
*Estructura de datos que contiene la información de la ciudad almacenada.*
- struct [Meteo](#)  
*Estructura de datos que contienen la información meteorológica de una ciudad en una fecha determinada.*

### Funciones

- void [printMeteoEntry](#) (struct [Meteo](#) meteoData)  
*La función [printMeteoEntry\(\)](#) muestra por pantalla la información de la entrada meteorológica recibida.*
- int [loadData](#) (const char \*filename, struct [City](#) \*\*cities)  
*La función [loadData\(\)](#) carga datos desde un fichero de texto en formato csv a un array de ciudades de tipo [City](#) (struct [City](#) \*). Para cada entrada de este array de ciudades, se encuentran los datos meteorológicos correspondientes a cada ciudad. Las ciudades son ordenadas por orden Alfabético para facilitar su búsqueda.*
- struct [Meteo](#) [customMeteoDataParser](#) (char \*line)  
*La función [customMeteoDataParser\(\)](#) recibe una string con los datos Meteorológicos en formato csv y los convierte a una estructura de datos de tipo struct [Meteo](#).*
- int [binarySearchCities](#) (char \*city, struct [City](#) \*cities, int numCities)  
*La función [binarySearchCities\(\)](#) realiza una búsqueda binaria en el array de ciudades.*
- int [binarySearchDates](#) (char \*date, struct [Meteo](#) \*meteoData, int numDataEntries)  
*La función [binarySearchCities\(\)](#) realiza una búsqueda binaria en el array de ciudades.*
- int [getForecast](#) (struct [City](#) \*cities, int numCities, struct [Meteo](#) \*\*forecast, char \*city, char \*date, int days)
- int [meteoDataToJSON](#) (struct [Meteo](#) \*meteoData, int meteoDataLength, char \*\*jsonData, char unit)

#### 4.7.1. Descripción detallada

En este fichero están declaradas aquellas funciones que están relacionadas con el manejo y procesamiento de los datos relacionados con la meteorología.

#### 4.7.2. Documentación de las funciones

##### 4.7.2.1. [binarySearchCities\(\)](#)

```
int binarySearchCities (
    char * city,
    struct City * cities,
    int numCities )
```

La función [binarySearchCities\(\)](#) realiza una búsqueda binaria en el array de ciudades.

**Parámetros**

<i>city</i>	Se trata de un string (char *) que contiene el nombre de la ciudad a buscar
<i>cities</i>	Se trata de un array de estructuras de tipo <a href="#">City</a> (struct <a href="#">City</a> *) que contiene las ciudades sobre las que se realizará la búsqueda
<i>numCities</i>	Se trata de un entero con el número de ciudades almacenadas

**Devuelve**

La función `binarySearchCities` devuelve la posición en el array de la ciudad si la ciudad es encontrada. Si no se ha encontrado devolverá -1.

**4.7.2.2. `binarySearchDates()`**

```
int binarySearchDates (
    char * date,
    struct Meteo * meteoData,
    int numDataEntries )
```

La función [binarySearchCities\(\)](#) realiza una búsqueda binaria en el array de ciudades.

**Parámetros**

<i>city</i>	Se trata de un puntero a una string (char *) que contiene el nombre de la ciudad a buscar
<i>cities</i>	Se trata de un array de estructuras de tipo <a href="#">City</a> (struct <a href="#">City</a> *) que contiene las ciudades sobre las que se realizará la búsqueda
<i>numCities</i>	Se trata de un entero con el número de ciudades almacenadas

**Devuelve**

La función `binarySearchCities` devuelve la posición en el array de la ciudad si la ciudad es encontrada. Si no se ha encontrado devolverá -1.

**4.7.2.3. `customMeteoDataParser()`**

```
struct Meteo customMeteoDataParser (
    char * line )
```

La función [customMeteoDataParser\(\)](#) recibe una string con los datos Meteorológicos en formato csv y los convierte a una estructura de datos de tipo struct [Meteo](#).

**Parámetros**

<i>line</i>	Se trata de un string (char *) que contiene la string con los datos que se quieren cargar.
-------------	--

**Devuelve**

La función `customMeteoDataParser()` devuelve una estructura de datos de tipo `Meteo` (struct `Meteo`) con los datos cargados del string line

**4.7.2.4. loadData()**

```
int loadData (
    const char * filename,
    struct City ** cities )
```

La función `loadData()` carga datos desde un fichero de texto en formato csv a un array de ciudades de tipo `City` (struct `City *`). Para cada entrada de este array de ciudades, se encuentran los datos meteorológicos correspondientes a cada ciudad. Las ciudades son ordenadas por orden Alfabético para facilitar su búsqueda.

**Parámetros**

<i>filename</i>	Se trata de un string (char *) que contiene el nombre del fichero de texto a cargar.
<i>cities</i>	Se trata de un puntero a un array de estructuras de tipo <code>City</code> (struct <code>City **</code> ) en el que se van a almacenar los datos cargados por ciudades

**Devuelve**

La función `loadData()` devuelve el número de elementos cargados si tiene éxito. En caso de error devolverá -1.

**Atención**

Es necesario liberar la memoria reservada en la función `loadData()` del parametro cities. Para ello liberaremos la información para cada entrada meteorológica para cada ciudad y posteriormente liberaremos la memoria reservada para cada ciudad.

**4.7.2.5. printMeteoEntry()**

```
void printMeteoEntry (
    struct Meteo meteoData )
```

La funcion `printMeteoEntry()` muestra por pantalla la información de la entrada meteorológica recibida.

**Parámetros**

<i>meteoData</i>	Se trata de una estructura de datos de tipo <code>Meteo</code> la cual contiene la información a mostrar por pantalla
------------------	---



# Índice alfabético

- binarySearchCities
  - meteo.c, [24](#)
  - meteo.h, [27](#)
- binarySearchDates
  - meteo.c, [24](#)
  - meteo.h, [28](#)
- celsiusToFahrenheit
  - helpers.c, [10](#)
  - helpers.h, [16](#)
- City, [5](#)
  - city, [5](#)
  - memory\_reserved, [5](#)
  - meteoData\_array, [5](#)
  - n\_meteoData\_entries, [6](#)
- city
  - City, [5](#)
  - Meteo, [6](#)
- cloudiness
  - Meteo, [6](#)
- compareDates
  - helpers.c, [10](#)
  - helpers.h, [16](#)
- compareString
  - helpers.c, [11](#)
  - helpers.h, [16](#)
- customMeteoDataParser
  - meteo.c, [24](#)
  - meteo.h, [28](#)
- date
  - Meteo, [7](#)
- fixDecimalNotation
  - helpers.c, [11](#)
  - helpers.h, [17](#)
- getIntFromUserInput
  - helpers.c, [11](#)
  - helpers.h, [17](#)
- getStringFromUserInput
  - helpers.c, [12](#)
  - helpers.h, [17](#)
- helpers.c, [9](#)
  - celsiusToFahrenheit, [10](#)
  - compareDates, [10](#)
  - compareString, [11](#)
  - fixDecimalNotation, [11](#)
  - getIntFromUserInput, [11](#)
  - getStringFromUserInput, [12](#)
  - replace, [12](#)
  - split, [12](#)
  - storeToFile, [13](#)
  - strCelsiusToStrFahrenheit, [13](#)
  - stringToFloat, [14](#)
  - trim, [14](#)
- helpers.h, [15](#)
  - celsiusToFahrenheit, [16](#)
  - compareDates, [16](#)
  - compareString, [16](#)
  - fixDecimalNotation, [17](#)
  - getIntFromUserInput, [17](#)
  - getStringFromUserInput, [17](#)
  - replace, [18](#)
  - split, [18](#)
  - storeToFile, [19](#)
  - strCelsiusToStrFahrenheit, [19](#)
  - stringToFloat, [20](#)
  - trim, [20](#)
- interfaces.c, [20](#)
  - showForecast, [21](#)
- interfaces.h, [21](#)
  - showForecast, [22](#)
- loadData
  - meteo.c, [25](#)
  - meteo.h, [29](#)
- main.c, [22](#)
- max\_temp
  - Meteo, [7](#)
- memory\_reserved
  - City, [5](#)
- Meteo, [6](#)
  - city, [6](#)
  - cloudiness, [6](#)
  - date, [7](#)
  - max\_temp, [7](#)
  - min\_temp, [7](#)
  - precipitations, [7](#)
- meteo.c, [23](#)
  - binarySearchCities, [24](#)
  - binarySearchDates, [24](#)
  - customMeteoDataParser, [24](#)
  - loadData, [25](#)
  - printMeteoEntry, [25](#)
- meteo.h, [27](#)
  - binarySearchCities, [27](#)

- binarySearchDates, [28](#)
- customMeteoDataParser, [28](#)
- loadData, [29](#)
- printMeteoEntry, [29](#)
- meteoData\_array
  - City, [5](#)
- min\_temp
  - Meteo, [7](#)
- n\_meteoData\_entries
  - City, [6](#)
- precipitations
  - Meteo, [7](#)
- printMeteoEntry
  - meteo.c, [25](#)
  - meteo.h, [29](#)
- replace
  - helpers.c, [12](#)
  - helpers.h, [18](#)
- showForecast
  - interfaces.c, [21](#)
  - interfaces.h, [22](#)
- split
  - helpers.c, [12](#)
  - helpers.h, [18](#)
- storeToFile
  - helpers.c, [13](#)
  - helpers.h, [19](#)
- strCelsiusToStrFahrenheit
  - helpers.c, [13](#)
  - helpers.h, [19](#)
- stringToFloat
  - helpers.c, [14](#)
  - helpers.h, [20](#)
- trim
  - helpers.c, [14](#)
  - helpers.h, [20](#)