# Guía de Ejercicios 6 - Strings

```
Advertencia

La resolución conjunta o grupal de los ejercicios aquí presentes no está permitida, excepto en
la medida en que puedas pedir ayuda a tus compañeros de clase y a otras personas, y siempre que
esa ayuda no se reduzca a que otro haga el trabajo por vos.

El código fuente entregado por un estudiante debe ser escrito en su totalidad por dicha persona.
```

**Condiciones de entrega:**

| ¿Qué se entrega? | ¿Qué no se entrega? |
|---|---|
| Archivos fuente/source (.c) | Archivos objeto (.o) |
| Archivos encabezado/header (.h) | Archivos ejecutables (programa, app, a.out, etc.) |
| Bibliotecas específicas (.a) | |

Se deben entregar los ejercicios en un archivo zip (usar template como ayuda para el formato).

**Importante:** Recordar validar **siempre** que no se reciben punteros `NULL`. En dicho caso, la función deberá retornar sin efectuar operación alguna y en caso de tener que retornar algún valor, devolverá el valor `-1`.

## Ejercicio 6.1

Implementar tu propia version de la función `strlen` de la biblioteca estándar `string`. Utilizar el siguiente prototipo:

```c
int strlen(const char* string);
```

## Ejercicio 6.2

Implementar tu propia version de la función `strcpy` de la biblioteca estándar `string`. Utilizar el siguiente prototipo:

```c
void strcpy(char* dest, const char* orig);
```

## Ejercicio 6.3

Implementar tu propia version de la función `strncpy` de la biblioteca estándar `string`. Utilizar el siguiente prototipo:

```c
void strncpy(char* dest, const char* orig, int n);
```

## Ejercicio 6.4

Implementar tu propia version de la función `strcmp` de la biblioteca estándar `string`. Utilizar el siguiente prototipo:

```c
int strcmp(const char* s1, const char* s2);
```

## Ejercicio 6.5

Implementar tu propia version de la función `strncmp` de la biblioteca estándar `string`. Utilizar el siguiente prototipo:

```
int strncmp(const char* s1, const char* s2, int n);
```

## Ejercicio 6.6

Implementar tu propia version de la función `strcat` de la biblioteca estándar `string`. Utilizar el siguiente prototipo:

```
void strcat(char* dest, const char* orig);
```

## Ejercicio 6.7

Implementar tu propia version de la función `strcasecmp` de la biblioteca estándar `string`. Utilizar el siguiente prototipo:

```
int strcasecmp(const char* s1, const char* s2);
```

## Ejercicio 6.8

Implementar tu propia version de la función `strncasecmp` de la biblioteca estándar `string`. Utilizar el siguiente prototipo:

```
int strncasecmp(const char* s1, const char* s2, int n);
```

## Ejercicio 6.9

Implementar una función que convierta todos los caracteres correspondientes a letras a mayúsculas. La función deberá retornar la cantidad de caracteres convertidos. Utilizar el siguiente prototipo:

```
int strupper(char* string);
```

## Ejercicio 6.10

Implementar una función que convierta todos los caracteres correspondientes a letras a minúsculas. La función deberá retornar la cantidad de caracteres convertidos. Utilizar el siguiente prototipo:

```
int strlower(char* string);
```

## Ejercicio 6.11

Implementar una función que reciba un string representando un número entero, y convierta su contenido al número que representa. Utilizar el siguiente prototipo:

```
int strtoi(const char* string);
```

## Ejercicio 6.12

Implementar una función que reciba un string representando un número real, y convierta su contenido al número que representa. Utilizar el siguiente prototipo:

```
float strtof(const char* string);
```

## Ejercicio 6.13

Implementar una función que invierta el casing, es decir, los caracteres que se encuentren en mayúsculas los convierta a minúsculas y los que están en minúsculas a mayúsculas. La función deberá retornar la cantidad de caracteres convertidos. Utilizar el siguiente prototipo:

```
int str_invertir_case(char* string);
```

## Ejercicio 6.14

Implementar una función que reciba un string e invierta el orden de sus caracteres. Utilizar el siguiente prototipo:

```
void str_invertir(char* string);
```

**Ejemplo:**

```
                ______________
               |              |
("batman") -->| str_invertir |--> "namtab"
               |______________|
```

## Ejercicio 6.15

Implementar una función que reciba un string y dos caracteres. La función deberá reemplazar las apariciones del primer caracter en el string por el segundo. Se deberá retornar la cantidad de veces que se reemplazó el caracter. Utilizar el siguiente prototipo:

```
int str_reemplazar(char* string, char caracter_original, char caracter_nuevo);
```

## Ejercicio 6.16

Implementar una función que reciba un string y devuelva `1` si el mismo es palíndromo, caso contrario devolverá `0`. Utilizar el siguiente prototipo:

```
int es_palindromo(const char* string);
```

**Ayuda:** Un palíndromo es una palabra o frase que se lee igual en un sentido que en otro. Si se trata de números en lugar de letras, se llama capicúa.

## Ejercicio 6.17

Implementar una función que cambie la extensión original del nombre de un archivo por una nueva. Utilizar el siguiente prototipo:

```
void cambiar_extension(char* nombre_archivo, const char* nueva_extension);
```

**Ejemplo:**

```
                           ____________________
                          |                    |
("archivo.txt", "ini") -->| cambiar_extension  |--> "archivo.ini"
                          |____________________|
```

## Ejercicio 6.18

Implementar una función que reciba un string y un caracter y la misma deberá retornar cuántas veces aparece dicho caracter en el string recibido. Utilizar el siguiente prototipo:

```
int contar_caracter(const char* string, char caracter);
```

## Ejercicio 6.19

Implementar una función que reciba un string y una palabra y la misma deberá retornar cuántas veces aparece dicha palabra en el string recibido. Utilizar el siguiente prototipo:

```
int contar_palabra(const char* string, const char* palabra);
```

**Importante:** No se deben distinguir mayúsculas y minúsculas, es decir, `"Casa"` y `"CASA"` son apariciones válidas de `"casa"`.

## Ejercicio 6.20

Un sistema de codificación muy famoso es el código Morse, desarrollado por Samuel Morse en 1832, para su uso en el sistema telegráfico. El mismo asigna una serie de puntos y rayas a cada letra del alfabeto, a cada dígito y a unos cuantos caracteres especiales.

Implementar una función que convierta un caracter en su equivalente en código Morse. La misma retornará `1` de haber sido posible la codificación o `0` en caso contrario. Utilizar el siguiente prototipo:

```
int a_morse(char caracter, char* codigo_morse);
```

## Ejercicio 6.21

Las fechas se expresan en varios formatos distintos, siendo dos de los más comunes `"21/09/98"` y `"21 de Septiembre de 1998"`.

Implementar una función que reciba una fecha en el primer formato (`"21/09/98"`) y devuelva la fecha en el segundo formato (`"21 de Septiembre de 1998"`). De poder realizar la conversión devolverá `1`, caso contrario devolverá `0`. Utilizar el siguiente prototipo:

```
int convertir_fecha_a_letras(const char* fecha_en_numeros, char* fecha_en_letras);
```

## Ejercicio 6.22

Las fechas se expresan en varios formatos distintos, siendo dos de los más comunes `"21 de Septiembre de 1998"` y `"21/09/98"`

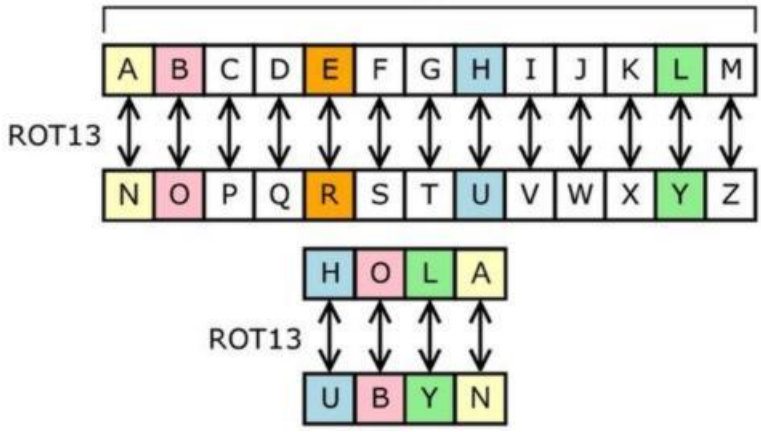
Implementar una función que reciba una fecha en el primer formato (`"21 de Septiembre de 1998"`) y devuelva la fecha en el segundo formato (`"21/09/98"`). De poder realizar la conversión devolverá `1`, caso contrario devolverá `0`. Utilizar el siguiente prototipo:

```
int convertir_fecha_a_numeros(const char* fecha_en_letras, char* fecha_en_numeros);
```

## Ejercicio 6.23

En criptografía, el cifrado César, también conocido como cifrado por desplazamiento o ROT13, es una de las técnicas de cifrado más simples y más usadas. Es un tipo de cifrado por sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto. Por ejemplo, con un desplazamiento de 3, la A sería sustituida por la D (situada 3 lugares a la derecha de la A), la B sería reemplazada por la E, etc. Este método debe su nombre a Julio César, que lo usaba para comunicarse con sus generales.

Entonces, para codificar un mensaje, simplemente se debe buscar cada letra de la línea del texto original y escribir la letra correspondiente en la línea codificada. Para decodificarlo se debe hacer lo contrario.




Implementar una función que codifique una frase usando el cifrado César. Utilizar el siguiente prototipo:

```
void cifrar_cesar(const char* frase_original, char* frase_codificada);
```

## Ejercicio 6.24

Implementar una función que reciba un string que contenga tres campos, en formato CSV. La función deberá retornar **por referencia** los tres strings extraídos. La función deberá retornar `ERROR_DE_PARSEO` (`1`) si ocurrió algún error, y `EXITO` (`0`) en caso contrario. Utilizar el siguiente prototipo:

```
int split_csv(const char* string_csv, char* campo_1, char* campo_2, char* campo_3);
```

**Ejemplo:**

```
                    ___________
                   |           |
("uno,dos,tres") -->| split_csv |-->("uno", "dos", "tres")
                   |___________|
```

## Ejercicio 6.25

Implementá una función que reciba tres strings y retorne **por referencia** un string en formato CSV uniendo todos los strings, agregando el caracter delimitador entre ellos (`','`). La función deberá retornar `ERROR_DE_PARSEO` (`1`) si ocurrió algún error, y `EXITO` (`0`) en caso contrario. Utilizar el siguiente prototipo:

```
int join_csv(const char* campo_1, const char* campo_2, const char* campo_3, char* string_csv);
```

**Ejemplo:**

```
                          _________
                         |         |
("uno", "dos", "tres")-->| join_csv |-->"uno,dos,tres"
                         |_________|
```

## Referencias

Algunos ejercicios fueron obtenidos y adaptados de:

- Guía de Trabajos Prácticos 2011 - Informática I - Departamento de Electrónica - UTN FRBA
- Cómo programar en C/C++ y Java - Harvey M. Deitel y Paul J. Deitel