Práctica 1
Reto 1

Implementación de un conjunto de funciones análogas a las de <string.h>

En esta práctica vamos a asumir 2 retos, por un lado vamos a implementar un **conjunto de funciones de tratamiento de cadenas** (reto 1), similares a las correspondientes de **<string.h>**, y por otro lado vamos a implementar un **gestor de pilas dinámico y genérico** (reto 2).

¿Qué conceptos trabajamos con el reto 1?:

- El uso de vectores de punteros/cadenas (examinar las inicializaciones del main() y paso de parámetros del código de test1.c, proporcionado con los ficheros de la práctica).
- El manejo de las funciones de <string.h> : strlen(), strcmp() [ver códigos ASCII], strcpy(), strncpy(), strcat(), strchr(). Probad los ejemplos con el CTutor para ver qué sucede con el '\0'.
- La utilización de ficheros cabecera (#include <*file*.h> *vs* #include "*file*.h").
- Y reforzaremos el manejo de arrays, punteros y strings.

¿En qué consiste detalladamente este reto?

Hay que crear una librería de funciones, denominada **my_lib.c**[1] que contenga la implementación de las siguientes funciones: **my_strlen()**[2], **my_strcmp(), my_strcpy(), my_strncpy(), my_strcat()** y **my_strchr()**. Tales funciones han de ser análogas a las respectivas de <string.h>.

Declaración de vuestras funciones	**Qué hace la correspondiente función de <string.h>**
size_t my_strlen(const char *str); **str**: apunta a la cadena de la cual se ha de encontrar la longitud.	La función strlen() calcula el nº de bytes de la cadena apuntada por **str**, sin incluir el carácter nulo de terminación '\0' .[3] Devuelve la longitud de la cadena apuntada por **str**. No devuelve error.

[1] No contendrá main(), sólo las definiciones de las funciones. Las declaraciones se encuentran en my_lib.h. Compilaremos usando el **Makefile** que os podéis descargar en el aula digital.
[2] Elegid una implementación de los ejemplos proporcionados.
[3] Observar la diferencia con sizeof().

int my_strcmp(const char *str1, const char *str2); • **str1**: apunta a la 1ª cadena a comparar. • **str2**: apunta a la 2ª cadena a comparar.	La función strcmp() compara las cadenas apuntadas por **str1** y **str2**. Devuelve un entero: • < 0 indica que str1 < str2. • > 0 indica que str2 < str1. • = 0 indica que str1 = str2. No se basa en la longitud de las cadenas sino que compara carácter a carácter y cuando encuentra uno diferente reporta el resultado calculado como la **resta de los códigos ASCII** de los caracteres diferentes.
char *my_strcpy(char *dest, const char *src); • **dest**: es el puntero a la cadena destino donde se copiará el contenido. • **src**: puntero a la cadena a copiar.	La función strcpy() copia la cadena apuntada por **src** (con el carácter de terminación '\0') en la memoria apuntada por **dest**. Devuelve el puntero **dest**. No devuelve error.
char *my_strncpy(char *dest, const char *src, size_t n); • **dest**: es el puntero a la cadena destino donde se copiará el contenido. • **src**: apunta a la cadena a copiar. • **n**: la cantidad de caracteres que se copiarán de la cadena apuntada por src.	La función strncpy() copia n caracteres de la cadena apuntada por **src** (con el carácter de terminación '\0') en la memoria apuntada por **dest**. Devuelve el puntero **dest**. No devuelve error. Si **n > strlen(src)**, el resto de la cadena apuntada por dest ha de ser rellenado con 0s. Si **n ≤ strlen(src)**, no se ha de añadir un carácter nulo a dest[n]. Recomendación: para probar la función, inicializar *dest rellenando con el carácter nulo[4] antes de llamar a strncpy(). [Ver Anexo con C Tutor]

[4] Se puede utilizar para ello la función `memset(dest, 0, sizeof(dest));`

char *my_strcat(char *dest, const char *src); • **dest**: es el puntero a la cadena destino. El espacio reservado para *dest ha de ser lo suficientemente grande para contener la cadena resultante de la concatenación. • **src**: apunta a la cadena a concatenar.	La función strcat() añade la cadena apuntada por **src** (terminada con el carácter nulo) a la cadena apuntada por **dest** (también terminada con el carácter nulo). El primer carácter de ***src** sobreescribe el carácter nulo de ***dest**. Devuelve el puntero **dest**. No devuelve error.
char *my_strchr(const char *str, int c); • **str**: apunta a la cadena a escanear. • **c**: es el carácter buscado.	La función strchr() escanea la cadena apuntada por **str** (terminada con el carácter nulo) buscando la primera ocurrencia del carácter **c**. Devuelve el puntero a la primera ocurrencia del carácter **c** en la cadena **str** o **NULL** si el carácter no se encuentra. No devuelve error. Observación: Si devolvéis la variable **str** hay que hacer un casting ya que se recibe como **const char *** y la función es de tipo **char ***: `return (char *)str;`

¿Qué recursos nos pueden ser útiles?

Documentos de apoyo:

- C Tutorial de Tutorialspoint (arrays, punteros, strings)
- A tutorial of pointers and arrays in C

Ficheros proporcionados (a descargar desde el aula virtual):

- **my_lib.h**: cabecera para incluir en **my_lib.c**
- **Makefile**[5]: para compilar conjuntamente vuestro **my_lib.c** con el **test1.c** y obtener

[5] Utilidad para compilar el conjunto de programas test1.c, my_lib.c y my_lib.h una vez que tengáis completas todas las funciones.
Situados en el directorio del Makefile se puede ejecutar lo siguiente:

- **make clean**: borra los objetos y ejecutables creados.

el ejecutable **test1**.

- **test1.c**: programa de testing que desde el main() llama a vuestras funciones de **my_lib.c** y compara su ejecución con las de **string.h**.

¿Y cómo podemos ir comprobando por nosotr@s mism@s que lo estamos logrando?

- Elaborando vuestras propias pruebas añadiendo **temporalmente**[6] una función principal en **my_lib.c** que haga llamadas a vuestras funciones con diferentes parámetros. Por ejemplo para probar my_strlen():

```
void main(){
 char cadena[]="Hola";
 printf("my_strlen(\"%s\"): %ld\n", cadena, my_strlen(cadena));
 return;
}
```

En caso de funciones que utilizan un puntero a una cadena fuente y un puntero a una cadena destino, para ésta última se debería haber reservado el espacio de memoria suficiente (estática o dinámica) para almacenar el resultado de la operación[7].

Observaciones con las **inicializaciones**:

```
void main(){
 char src[11]="programa";
 char dest[11]="compilador";
 //Reserva 11 posiciones de memoria (stack) y el contenido es modificable
 printf("%s\n", strcpy(dest,src));
 return;
}
```

```
void main(){
 char src[11]="programa";
 char *dest="compilador";
 //El contenido apuntado por dest[8]  NO es modificable, se considera un literal (constante).
 //Se almacena en una zona de memoria que es de sólo lectura (ni en la pila ni en el heap)
 //Ocurrirá una violación de segmento[9] con la siguiente instrucción:
 printf("%s\n", strcpy(dest,src));
```

- **make (all)**: compila el conjunto de librería y programas.
- **make test1**: compila librería y test1.

[6] En tal caso hay que eliminar luego ese main() para entregar el programa.
[7] En esta práctica se encarga de hacerlo el main() de test1.c
[8] Las cadenas de caracteres constantes se almacenan en una sección especial **.rodata** (*read only data*) que, en algunas implementaciones, podría formar parte del **.text**.
[9] Más info del problema en http://polyglotprog.blogspot.com/2011/09/what-is-reason-for-sigsegv-bad.html

```
return;
}
```

```
void main(){
 char src[11]="programa";

 char *dest= malloc(11);
//Reserva 11 posiciones de memoria (heap) y el contenido es modificable.
 strcpy(dest,"compilador");

 printf("%s\n", strcpy(dest,src));
 return;
}
```

- Ejecutando el **test1** (tenéis que descargar **test1.c** en el aula digital, en la carpeta "Ficheros fuente práctica1", junto al **Makefile** y **my_lib.h**) y contrastando vuestros resultados con éstos:

```
$ make test1
$ ./test1

****************************************
Testeando my_strlen() frente a strlen()
****************************************
*str: "programa"
strlen(str) = 8
my_strlen(str) = 8

*str: "compilador"
strlen(str) = 10
my_strlen(str) = 10

*str: "depurador"
strlen(str) = 9
my_strlen(str) = 9

test_strlen passed :-)

****************************************
Testeando my_strcmp() frente a strcmp()
****************************************
*str1: "programa", *str2: "depurador"
strcmp(str1, str2) = 12
my_strcmp(str1, str2) = 12
```

```
*str1: "programa", *str2: "compilador"
strcmp(str1, str2) = 13
my_strcmp(str1, str2) = 13

*str1: "programa", *str2: "programa"
strcmp(str1, str2) = 0
my_strcmp(str1, str2) = 0

*str1: "compilador", *str2: "depurador"
strcmp(str1, str2) = -1
my_strcmp(str1, str2) = -1

*str1: "compilador", *str2: "compilador"
strcmp(str1, str2) = 0
my_strcmp(str1, str2) = 0

*str1: "compilador", *str2: "programa"
strcmp(str1, str2) = -13
my_strcmp(str1, str2) = -13

*str1: "depurador", *str2: "depurador"
strcmp(str1, str2) = 0
my_strcmp(str1, str2) = 0

*str1: "depurador", *str2: "compilador"
strcmp(str1, str2) = 1
my_strcmp(str1, str2) = 1

*str1: "depurador", *str2: "programa"
strcmp(str1, str2) = -12
my_strcmp(str1, str2) = -12

test_strcmp passed :-)

****************************************
Testeando my_strcpy() frente a strcpy()
****************************************
*dest: "programa", *src: "compilador"
strcpy(dest, src) = compilador
my_strcpy(dest, src) = compilador

*dest: "compilador", *src: "depurador"
strcpy(dest, src) = depurador
my_strcpy(dest, src) = depurador
```

```
*dest: "depurador", *src: "programa"
strcpy(dest, src) = programa
my_strcpy(dest, src) = programa

test_strcpy passed :-)

************************************************
Testeando my_strncpy() frente a strncpy(). Test 1
************************************************
*dest: "programa", *src: "compilador", n = 1
strncpy(dest, src, 1) = crograma
i: 0, strlen(src): 10, dest:crograma
my_strncpy(dest, src, 1) = crograma

*dest: "programa", *src: "compilador", n = 4
strncpy(dest, src, 4) = comprama
i: 0, strlen(src): 10, dest:crograma
i: 1, strlen(src): 10, dest:coograma
i: 2, strlen(src): 10, dest:comgrama
i: 3, strlen(src): 10, dest:comprama
my_strncpy(dest, src, 4) = comprama

*dest: "programa", *src: "compilador", n = 7
strncpy(dest, src, 7) = compilaa
i: 0, strlen(src): 10, dest:crograma
i: 1, strlen(src): 10, dest:coograma
i: 2, strlen(src): 10, dest:comgrama
i: 3, strlen(src): 10, dest:comprama
i: 4, strlen(src): 10, dest:compiama
i: 5, strlen(src): 10, dest:compilma
i: 6, strlen(src): 10, dest:compilaa
my_strncpy(dest, src, 7) = compilaa

*dest: "programa", *src: "compilador", n = 10
strncpy(dest, src, 10) = compilador
i: 0, strlen(src): 10, dest:crograma
i: 1, strlen(src): 10, dest:coograma
i: 2, strlen(src): 10, dest:comgrama
i: 3, strlen(src): 10, dest:comprama
i: 4, strlen(src): 10, dest:compiama
i: 5, strlen(src): 10, dest:compilma
i: 6, strlen(src): 10, dest:compilaa
i: 7, strlen(src): 10, dest:compilad
i: 8, strlen(src): 10, dest:compilado
i: 9, strlen(src): 10, dest:compilador
```

```
my_strncpy(dest, src, 10) = compilador

test_strncpy passed :-)

************************************************
Testeando my_strncpy() frente a strncpy(). Test 2
************************************************
*dest: "compilador", *src: "programa", n = 1
strncpy(dest, src, 1) = pompilador
i: 0, strlen(src): 8, dest:pompilador
my_strncpy(dest, src, 1) = pompilador

*dest: "compilador", *src: "programa", n = 4
strncpy(dest, src, 4) = progilador
i: 0, strlen(src): 8, dest:pompilador
i: 1, strlen(src): 8, dest:prmpilador
i: 2, strlen(src): 8, dest:propilador
i: 3, strlen(src): 8, dest:progilador
my_strncpy(dest, src, 4) = progilador

*dest: "compilador", *src: "programa", n = 7
strncpy(dest, src, 7) = programdor
i: 0, strlen(src): 8, dest:pompilador
i: 1, strlen(src): 8, dest:prmpilador
i: 2, strlen(src): 8, dest:propilador
i: 3, strlen(src): 8, dest:progilador
i: 4, strlen(src): 8, dest:progrlador
i: 5, strlen(src): 8, dest:prograador
i: 6, strlen(src): 8, dest:programdor
my_strncpy(dest, src, 7) = programdor

*dest: "compilador", *src: "programa", n = 10
strncpy(dest, src, 10) = programa
i: 0, strlen(src): 8, dest:pompilador
i: 1, strlen(src): 8, dest:prmpilador
i: 2, strlen(src): 8, dest:propilador
i: 3, strlen(src): 8, dest:progilador
i: 4, strlen(src): 8, dest:progrlador
i: 5, strlen(src): 8, dest:prograador
i: 6, strlen(src): 8, dest:programdor
i: 7, strlen(src): 8, dest:programaor
my_strncpy(dest, src, 10) = programa

test_strncpy passed :-)
```

```
**************************************
Testeando my_strcat() frente a strcat()
**************************************
*dest: "programa", *src: "compilador"
strcat(dest, src) = programacompilador
my_strcat(dest, src) = programacompilador

*dest: "compilador", *src: "depurador"
strcat(dest, src) = compiladordepurador
my_strcat(dest, src) = compiladordepurador

*dest: "depurador", *src: "programa"
strcat(dest, src) = depuradorprograma
my_strcat(dest, src) = depuradorprograma

test_strcat passed :-)

**************************************
Testeando my_strchr() frente a strchr()
**************************************
*srt: "programa", char: 'm'
strchr(str, char) = ma
my_strchr(str, char) = ma

*srt: "compilador", char: 'm'
strchr(str, char) = mpilador
my_strchr(str, char) = mpilador

*srt: "depurador", char: 'm'
strchr(str, char) = (null)
my_strchr(str, char) = (null)


test_strchr passed :-)

$ make clean
```

Observaciones

- Uso de **const** en los parámetros:
  El calificador const le permite al programador informarle al compilador que no se debe modificar el valor de una variable en particular.
- Uso de **make clean**

En realidad no hace falta hacer **make clean** cada vez que vamos a compilar una parte pues el make es inteligente y sabe qué archivos compilar si se han hecho modificaciones. Pero si trasladáis vuestra carpeta de un ordenador a otro con distinta arquitectura y no hacéis **make clean** es posible mezclar código de una arquitectura con otra y os dará error. Por consiguiente, en la primera compilación en un ordenador siempre, la primera vez, hacer **make clean,** luego, con el **make** adecuado es suficiente.

Anexo: Pruebas con C Tutor

```c
#include <stdio.h>
#include <string.h>
int main () {
 char src[12], dest[12] ;

 //caso 0: n < strlen(src), dest vacío
 strcpy(src,  "depurador");
 //dest requiere inicialización previa a strncpy()
 //ya que si no dest queda sin '\0'
 memset(dest, '\0', sizeof(dest));
 strncpy(dest, src, 1);
 printf("Resultado: %s\n", dest);
 return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
int main () {
 char src[12], dest[12] ;

 //caso 1: n < strlen(src) , strlen(dest) > strlen(src)
 memset(dest, '\0', sizeof(dest));
 strcpy(dest, "compilador");
 strcpy (src, "programa");
 printf("%s\n",strncpy(dest, src, 3));
 /* No se añade '\0' tras copiar 3 caracteres de src,
  dest conserva su '\0' */
 return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
int main () {
 char src[12], dest[12] ;

 //caso 2: n = strlen(src) , strlen(dest) > strlen(src)
 memset(dest, '\0', sizeof(dest));
 strcpy(dest, "compilador");
 strcpy (src, "programa");
 printf("%s\n",strncpy(dest, src, strlen(src)));
 /* No se añade '\0' tras copiar todos los caracteres de src menos '\0',
  dest conserva su '\0' */
 return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
int main () {
 char src[12], dest[12] ;

 //caso 3: n > strlen(src) , strlen(dest) > strlen(src)
 memset(dest, '\0', sizeof(dest));
 strcpy(dest, "compilador");
 strcpy (src, "programa");
 printf("%s\n",strncpy(dest, src, strlen(dest)));
 /* Se añade un '\0' al finalizar la copia de src
  que machaca lo que queda de dest */
 return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
int main () {
 char src[12], dest[12] ;

 //caso 4: n < strlen(src), strlen(dest) < strlen(src)
 memset(dest, '\0', sizeof(dest));
 strcpy (dest, "programa");
 strcpy(src, "compilador");
 printf("%s\n",strncpy(dest, src, strlen(src)-5));
 /* No se añade '\0' tras copiar 5 caracteres de src,
  dest conserva su '\0' */
 return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
int main () {
 char src[12], dest[12] ;

 //caso 5: n = strlen(src) , strlen(dest) < strlen(src)
 memset(dest, '\0', sizeof(dest));
 strcpy (dest, "programa");
 strcpy(src, "compilador");
 printf("%s\n",strncpy(dest, src, strlen(src)));
 /* No se añade '\0' tras copiar todos los caracteres de src menos '\0',
  dest conserva su '\0' */
 return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
int main () {
 char src[12], dest[12] ;

 //caso 6: n > strlen(src), strlen(dest) < strlen(src)
 memset(dest, '\0', sizeof(dest));
 strcpy (dest, "programa");
 strcpy(src, "compilador");
 printf("%s\n",strncpy(dest, src, strlen(src)+2));
 return 0;
}
```