ft_substr.c

Prototipo: char *ft_substr(char const *s, unsigned int start, size_t len)

s = String a enviar y revisar.
start = posición inicial de la cadena.
len = longitud de la cadena a devolver.

Descripción: Devuelve una subcadena con la cadena que se le ha proporcionado. La posición [start] es donde debe iniciar y la posición [len] es donde tiene que terminar.

Char *ft_substr(char const *s, unsigned int start, size_t len) {	Prototipo.
size_t i; char *str;	Variable para contar posiciones. Variable para la subcadena.
i = ft_strlen(s);	Vamos a saber el largo de la cadena [s].
if (!s) return (NULL);	Sí [s] está vacío. Entonces, devolverá NULL. (End function).
if (start > i) return (NULL);	Si [start] es mayor que [i], eso significa que la posición a empezar es mayor que el largo de la cadena enviada. Entonces, devolverá NULL. (End function)
if (start + len > i) return (NULL);	Si [start] más [len] es mayor que [i], eso significa que la posición a terminar es mayor que el largo de la cadena enviada. Entonces, devolverá NULL. (End function)

str = (char *)malloc(sizeof(*s) * (len + 1));
if (!str)
return (NULL);
	Reservamos la memoria en el Heap y comprobamos si tenemos un resultado -true- de reserva, De lo contrario, devolvermos NULL. (End function).

ft_memcpy(str, s + start, len);	Enviamos a ft_memcpy.c para generar en la memoria reservada la substring.
str[len] = '\0';	A la última posición de la memoría reservada le ponemos el Nulo para finalizar la string.
return (str); }	Devolvemos el resultado que tenemos en la nueva memoria reservada. (End function).

Main:

```
int main()
{
	char  s[] = "Me gusta programar en 42 Barcelona";
	int start = 22;
	int len = 13;

	printf("Resultado: %s\n", ft_substr(s, start, len));
	return 0;
}
```

Devuelve:

“42 Barcelona”

ft_strjoin.c

Protopito: char *ft_strjoin(char const *s1, char const *s2)

s1 = String número 1.
s2 = String número 2.

Descripción: El objetivo de esta función es lograr concatenar las dos 'string' en una sola cadena reservada con Malloc.

static int ft_while(char const *src, char *str, size_t j) {	Prototipo de la función auxiliar.
int i; i = 0;	Variable para contar posiciones; Iniciamos la posición a 0.

while (src[i])
{
str[j] = src[i];
i++;
j++;
}
	Mientras que [src] que es la string s1 o s2, no llegue al final del todo.
	En la reserva será igual que [src].
	Sumamos la posición i;
	Sumamos la posición x;

return (j); }	Devolemos la posición actual de x que será la marca para la siguiente cadena. (End function).
char *ft_strjoin(char const *s1, char const *s2) {	Prototipo de la función
char *str; int len_s1; int len_s2; int j;	Variable para la cadena (heap) Variable largo de la cadena [s1] Variable largo de la cadena [s2] Variable posición cadena (heap)
len_s1 = ft_strlen(s1); len_s2 = ft_strlen(s2);	Largo de [s1] Largo de [s2]

str = (char *)malloc(sizeof(char) * (len_s1 + len_s2 + 1));
if (!str)
{
return (NULL);
}
	Reservamos memoria en el Heap sumando el largo de [s1] y [s2] más 1 para para finalizar la cadena.
	Si no tenemos suficiente memoria, devolvemos NULL (End function)

j = 0;
j = ft_while(s1, str, j);
j = ft_while(s2, str, j);
	Iniciamos [j] a la posición 0;
	Mandamos la [s1] a la función auxiliar que nos devolverá la posición.
	Mandamos la [s2] con la posición recibida anteriormente a la función auxiliar.

str[j] = '\0'; return (str); }	Con la siguiente posición recibida escribimos el carácter NULO para finalizar la cadena. Devolvemos la cadena copiada en el Heap.

Main:

```
int main()
{
  char s1[] = "Me gusta programar en ";
  char set[] = "42 Barcelona";
  printf("Resultado: %s\n", ft_strjoin(s1, s2));
  return 0;
}
```

Devuelve:

"Me gusta programar en 42 Barcelona\0"

ft_strtrim.c

Prototipo: char *ft_strtrim(char const *s1, char const *set)

s1 = String a evaluar;
set = Carácteres a descartar;

Descripción: El objetivo de esta función es buscar en la cadena [s1], tanto por delante, como por detrás, aquellos caracteres que queremos descartar [set] hasta que encontremos un carácter no perteneciente. El resultado nos mostrará los caracteres que no se han descartado.

char *ft_strtrim(char const *s1, char const *set) {	Prototipo de la función
char *str; size_t i;	Variable donde reservaremos la nueva cadena. Variable largo de [s1]
if (!s1 || !set) return (NULL);	Si [s1] o [set] están vacíos Entonces, devolvemos NULL (End function)
while (*s1 && ft_strchr(set, *s1)) s1++;	Mientras que la posición [s1] y la comparación en ft_strchr.c sea -true-, sumaremos una posición a [s1].
i = strlen(s1);	A la [i] le daremos el valor del largo de la cadena [s1].

while (i && strchr(set, s1[i]))
{
i--;
}
	Mientras que [i] y la comparación en ft_strchr.c sea -true-, restaremos una posción a [i].

str = ft_substr (s1, 0, i + 1);	Mandaremos la nueva cadena a ft_substr.c donde reservará un espacio de memoria (Heap).
return (str); }	Devolveremos el resultado de la nueva cadena reservada en el Heap.

Main:

```
int main() {

  char s1[] = "hhhrolllrrwwa";
  char set[] = "hola";
  printf("Resultado: %s\n", ft_strtrim(s1, set));
  return 0;
}
```

Devuelve:

"rolllrrww"

ft_split.c

Prototipo: char **ft_split(char const *s, char c)

s = Cadena a separar.
c = Carácter delimitador.

Descripción: El objetivo es separar la cadena en varias subcadenas utilizando como referencia el carácter delimitador.

`static size_t ft_count_del(char const *s, char c)` `{`	Función auxiliar para contar los delimitadores.
`int i;` `int del;`	Variable para contar posiciones Variable para contar delimitadores
`i = 0;` `del = 0;`	Iniciamos la posición [i] a 0 Iniciamos la [del] a 0

`while (s && s[i])`
`{`
`if (s[i] != c)`
`{`
`while (s[i] != c && s[i])`
`{`
`i++;`
`}`
`del++;`
`}`
`else`
`{`
`i++;`
`}`
`}`
	Mientras que la cadena [s] y la posición [i] de la cadena [s] sea -true-
	Si la posición [i] de la cadena [s] no es igual al delimitador enviado [c]
	Mientras que la posición [i] de la cadena [s] no sea igual al delimitador enviado [c] y la posición [i] de la cadena [s] sea -true- (no es NUL)
	Sumaremos una posición.
	
	Si no, sumaremos un delimitador.
	
	Entonces
	Sumaremos una posición.

`return (del);` `}`	Devuelve un número que son la cantidad de delimitadores
`static char **ft_free(char **str)` `{`	Función auxiliar para liberar malloc
`size_t i;`	Variable para contar
`i = 0;`	Iniciamos la [i] a 0

`while (str[i])`
`{`
`free (str[i]);`
`i++;`
`}`
	Mientras la posición [i] de la cadena sea -true- (no es NUL)
	Liberamos la posición (Subreserva Heap)
	Sumamos una posición

`free(str);`	Liberamos la primera reserve Heap
`return (NULL);` `}`	Devolvemos NULL (End function)
`char **ft_split(char const *s, char c)` `{`	Prototipo de la función

`char **srcs;`
`size_t i;`
`size_t count;`
`size_t pos;`
	Variable donde reservaremos la nueva cadena
	Variable contador posición delimitadores
	Variable contador cadena
	Variable contador posición actual cadena.

`i = 0;` `count = 0;`	Iniciamos [i] a 0 Iniciamos [count] a 0

`srcs = (char **)malloc(sizeof(char *) * (ft_count_del(s, c) + 1));`
`if (!srcs)`
`return (NULL);`
	Reservarmos memoria en el Heap con la devolución de la cantidad de delimitadores + 1 para NUL fin de cadena.
	Si no hay memoria retornará NULL (End function)

`while (i < ft_count_del(s, c) && s[count] != '\0')`
`{`
`  while (s[count] == c)`
`  count++;`
`  pos = count;`
`  while (s[count] != c && s[count] != '\0')`
`  count++;`
`  srcs[i] = ft_substr(s, pos, count - pos);`
`  if (srcs[i] == NULL)`
`  return (ft_free(srcs));`
`  i++;`
`}`
	-Mientras [i] sea menor que la cantidad de delimitadores y la posición [count] de la cadena [s] no sea NUL
	-Mientras la posición[count] de la cadena [s] sea igual al delimitador [c], sumaremos una posición [count]
	Iniciamos posición actual [pos] igual a la posición de la cadena [count]
	-Mientras la posición [count] de la cadena [s] no sea igual al delimitador [c] y la posición [count] de la cadena [s] no sea NUL
	Sumaremos una posición [count]
	-Mandaremos en la posición [i] de la reserva de Heap a ft_substr.c con los parámetros de la cadena entera, posición actual [pos] y posición de la cadena [count] menos la posición actual [pos].
	-Si la posición [i] de la reserva es igual a NULL mandaremos a liberar la reserva d memoria (End function)
	-Sumaremos una posición a la reserva de memoria.

`srcs[i] = '\0';`	La última posición de la reserva de Heap será NULL
`return (srcs);` `}`	Devolveremos todas las reservas de la reserva de memoria (delimitadores) con un array en el main.

Main:

```
int main()
{
  char const s[] = "JAN,FEB,MARCH,APRIL,MAY";
  char c[] = ",";
  int i = 0;
  printf("s: %s\nc:", s, c);
  array = ft_split(s, c);
  while (i < 5)
  {
       printf("Resultado: [%d] = %s\n", i, array[i]);
       i++;
  }
  return 0;
}
```

Devuelve:

```
"s: JAN,FEB,MARCH,APRIL,MAY"
"c: ,"
"Resultado: [0] = JAN"
"Resultado: [1] = FEB"
"Resultado: [2] = MARCH"
"Resultado: [3] = APRIL"
"Resultado: [4] = MAY"
```

ft_itoa.c

Prototipo: char *ft_itoa(int n)

n = Número a enviar.

Descripción: Convierte número integer en número char.

static int ft_len_number(int n) {	Prototipo auxiliar para contar el largo del número.
int i;	Variable para contar posiciones
i = 0;	Inicializamos la posición a 0

if (n <= 0)
{
i++;
}
	Si el número enviado [n] es menor o igual a 0.
	
	Sumamos una posición [i]

while (n != 0)
{
n = n / 10;
i++;
}
	Mientras el número enviado [n] no sea igual a 0.
	
	El número enviado lo dividiremos entre 10 para descartar la última posición
	Sumamos una posición [i]

return (i); }	Devolvemos el total de la suma de [i]
char *ft_itoa(int n) {	Prototipo de la función
long num; int len; char *str;	Variable para almacenar el número recibido Variable para almacenar el total de la cadena Variable para guardar en memoria el número recibido
num = n;	[num] será igual al número que hemos recibido como parámetro.
len = ft_len_number(n);	[len] será igual al largo del número recibido como parámetro.

str = (char *)malloc(sizeof(char) * len + 1);
if (!str)
return (NULL);
	Reservamos memoria en el Heap con el largo del [len] más 1 para el NUL. Si no hay espació de memoria devolverá NULL (End function)

str[len] = '\0';	Escribimos al final de la cadena [str] el símbolo NUL
len--;	Restamos una posición a [len]
if (num == 0) str[0] = 48;	Si [num] es igual a cero En la posición “0” de la cadena escribiremos “48” que es igual a “0” en la tabla ASCII

if (num < 0)
{
str[0] = 45;
num = -num;
}
	Si [num] es menor a “0”
	En la posición “0” de la cadena escribiremos 45 que es igual a “-” en la tabla ASCII
	Convertiremos [num] en un número positivo.

while (num > 0)
{
str[len] = (num % 10) + 48;
num = num / 10;
len--;
}
	Mientras que [num] sea mayor a 0
	EN la posición [len] será igual al resto de la división entre “10” + 48 para convertir el número en char.
	Dividiremos el número entre 10 para descartar la última posición.
	Restaremos una posición a [len]

return (str); }	Devolveremos los datos que hemos ido reservando en el Heap.

Main:

```
int main()
{
	printf("Resultado: %s\n", ft_itoa(1234567890));
	return 0;
}
```

Devuelve:

“1234567890”

ft_strmapi.c

Prototipo: char *ft_strmapi(char const *s, char (*f)(unsigned int, char))

s = cadena a enviar.
*f = función a realizar.

Descripción: El objetivo de esta función es enviar cada carácter de la cadena para que se aplique la función creando una nueva cadena con el resultado del uso sucesivo de la [f]

char *ft_strmapi(char const *s, char (*f)(unsigned int, char)) {	Prototipo de la función
char *src; int i; int len;	Variable para la reserva de la nueva cadena Variable para contar posiciones Variable para saber el largo.
i = 0;	Iniciamos la posición a '0'

if (!s)
{
return (NULL);
}
	Si la cadena enviada [s] está vacía.
	Entonces, devolvemos NULL (End function)

len = ft_strlen(s);	Contamos el largo de la cadena enviada [s]

src = (char *)malloc(sizeof(char) * (len + 1));
if (!src)
{
return (NULL);
}
	Reservamos en el Heap con el largo [count] de la cadena enviada más 1 para finalizar la nueva cadena.
	Si no hay espacio en el Heap devolverá NULL (End function)

while (i < len)
{
src[i] = f(i, s[i]);
i++;
}
	Mientras la posición [i] sea menor al largo de la cadena [count]
	Entonces, la primera posición del Heap será igual a la función estipulada
	Aumentamos una posición [i]

src[i] = '\0';	Al final añadiremos NUL a la última posición del Heap
return (src); }	Devolveremos la nueva cadena reservada en el Heap

Main:

```
int     main()
{
        char a[] = "Me gusta programar en 42 Barcelona"

        printf("Resultado: %s\n", ft_strmapi(a, to_upper));
        return (0);
}
```

Devuelve:

"ME GUSTA PROGRAMAR EN 42 BARCELONA"

Para la función, tenemos que reciclar una función según los parámetros asignados en el prototipo. En el caso de mi ejemplo, he realizado una función donde cogerá la array enviada y los convertirá en mayúsculas, una a una.

Función auxiliar:

```
void    to_upper(unsigned int i, char a)
{
        if(*a >= 'a' && *a <= 'z')
                *a = *a - 32;
}
```

ft_striteri.c

Prototipo: void ft_striteri(char *s, void (*f)(unsigned int, char*))

s = cadena a enviar
f* = función a enviar

Descripción: El objetivo de esta función es enviar cada carácter de la cadena para que se aplique la función.

void ft_striteri(char *s, void (*f)(unsigned int, char*)) {	Prototipo de la función
int i;	Variable para contar posiciones
i = 0;	Iniciamos la posición a '0'

if (s != NULL && f != NULL)
{
while (s[i])
{
f(i, s + i);
i++;
}
}
}
	Si la cadena [s] y la función a realizar [f] están vacías (End function)
	Mientras la cadena [s] no llegue al final (NUL)
	
	Pasaremos la posición a la función estipulada.
	Sumaremos una posición más.

Main:

```
int    main()
{
       char a[] = "Me gusta programar en 42 Barcelona";

       printf("Primer resultado: %s.\n" , a);
       ft_striteri(a, to_upper);
       printf("El resultado final es: %s.", a);
       return (0);
}
```

Devuelve:

"ME GUSTA PROGRAMAR EN 42 BARCELONA"

Para la función, tenemos que reciclar una función según los parámetros asignados en el prototipo. En el caso de mi ejemplo, he realizado una función donde cogerá la array enviada y los convertirá en mayúsculas, una a una.

Función auxiliar:

```
void   to_upper(unsigned int i, char a)
{
       if(*a >= 'a' && *a <= 'z')
              *a = *a – 32;
}
```

ft_putchar_fd.c

Prototipo: void ft_putchar_fd(char c, int fd)

c = El carácter a enviar
fd = File descriptor sobre el que escribir.

Descripción: El objetivo de la función es mandar el carácter que enviamos [c] a un archivo [fd]. Este archivo puede estar creado o se creará automáticamente si no existe en el folder.

void ft_putchar_fd(char c, int fd) {	Prototipo de la función
write (fd, &c, 1);	Función para escribir en el file descriptor [fd] enviando la dirección del puntero.
}	

Main:

```
int main(void)
{
	int fd;
	fd = open("test_putchard", O_RDWR | O_CREAT);
	ft_putchar_fd('T', fd);
	return (0);
}
```

Devuelve:

"test putchard"
O_RDWR: Read and write.
O_CREAT: Si el archive no existe se crea.

ft_putstr_fd.c

Prototipo: void ft_putstr_fd(char *s, int fd)

s = La cadena a enviar.
fd = File descriptor sobre el que escribir.

Descripción: El objetivo de la función es mandar la cadena que enviamos [c] a un archivo [fd]. Este archivo puede estar creado o se creará automáticamente si no existe en el folder.

void ft_putstr_fd(char *s, int fd) {	Prototipo de la función

if (!s)
{
return ;
}
	Sí no existe una cadena enviada.
	
	Entonces se termina la función (End function)

while (*s)
{
write (fd, s, 1);
s++;
}
}
	Mientras la cadena no llegue al final (NUL)
	
	Función para escribir en el file descriptor [fd] enviando la posición del puntero.
	Se sumará una posición a la cadena.

Main:

```
int main(void)
{
	int fd;
	char src[] = "Me gusta programar en 42 Barcelona";

	fd = open("test_putchard", O_RDWR | O_CREAT);
	ft_putstr_fd(src, fd);
	return (0);
}
```

Devuelve:

"Me gusta programar en 42 Barcelona"
O_RDWR: Read and write.
O_CREAT: Si el archive no existe se crea.

ft_putendl_fd.c

Prototipo: void ft_putendl_fd(char *s, int fd)

s = La cadena a enviar.
fd = El file descriptor sobre el que escribir.

Descripción: El objetivo de la función es escribir una cadena de caracteres enviada [s] en el file descriptor [fd]. Una vez finalizado añadirá un salto de línea al final.

void {	ft_putendl_fd(char *s, int fd)	Prototipo de la función

	if (!s)
	{
	return ;
	}
		Si no existe una cadena enviada.
		
		Entonces se termina la función (End function)

	while (*s)
	{
	write (fd, s, 1);
	s++;
	}
		Mientras la cadena no llegue al final (NUL)
		
		Función para escribir en el file descriptor [fd] enviando la posición del puntero.
		Se sumará una posición a la cadena.

} }	write (fd, "\n", 1);	Función para escribir el salto del línea.

Main:

```
int main(void)
{
	int fd;
	char src[] = "Me gusta programar en 42 Barcelona";

	fd = open("test_putchard", O_RDWR | O_CREAT);
	ft_putendl_fd(src, fd);
	return (0);
}
```

Devuelve:

“Me gusta programar en 42 Barcelona”
O_RDWWR: Read and write.
O_CREAT: Si el archive no existe se crea.

ft_putnbr_fd.c

Prototipo: void ft_putnbr_fd(int n, int fd)

n = El número a enviar.
fd = El file descriptor sobre el que escribir.

Descripción: El objetivo de la función es escribir un número enviado [n] en el file descriptor [fd].

static void ft_putchar(int n, int fd) {	Prototipo de la función auxiliar para convertir el número en char
char c;	Variable donde se almacenará el número como char [c]
c = n + '0';	Conversión del número [n] en char [c]
write (fd, &c, 1); }	Función para escribir en el file descriptor [fd] el número enviado ya convertido en char [c]. Mandamos la dirección.
void ft_putnbr_fd(int n, int fd) {	Prototipo de la función

if (n == -2147483648)
{
write (fd, "-2147483648", 11);
}
	Si el número [n] enviado es igual al número menor máximo de integer.
	Función para escribir en el file descriptor [fd].

else if (n < 0)
{
n = -n;
write (fd, "-", 1);
ft_putnbr_fd(n, fd);
}
	Entonces si, el número [n] enviado es menor a '0'.
	
	Convertimos el número [n] en positivo
	Función para escribir el negativo
	Reiterativa a la función con el número [n] ya en positivo.

else if (n > 9)
{
ft_putnbr_fd(n / 10, fd);
ft_putnbr_fd(n % 10, fd);
}
	Entonces si, el número [n]
	
	Reiterativa a la función con el número [n] menos el último dígito dividiéndolo entre 10
	Reiterativa a la función con el número [n] que es el residuo de la división entre 10

else
{
ft_putchar(n, fd);
}
}	

Main:

```
int main(void)
{
	int fd;

	fd = open("test_putchard", O_RDWR | O_CREAT);
	ft_putnbr_fd(4546, fd);
	return (0);
}
```

Devuelve:

"4546"
O_RDWWR: Read and write.
O_CREAT: Si el archive no existe se crea.

ft_isalpha.c

Prototipo: int ft_isalpha(int c)

c = El carácter a revisar.

Descripción: El objetivo de la función es comprobar si el carácter [c] enviado es alfabético.

int ft_isalpha(int c) {	Prototipo de la función

if ((c >= 'A' && c <= 'Z') || (c >= 'a' || c <= 'z'))
{
return (1);
}
	Si el carácter enviado [c] es mayúscula y es minúscula
	
	Entonces retornará 0 (End function)

return (0); }	Si no, retornará 1 (End function)

Main:

```
int main()
{

	printf("Resultado: %s\n", ft_isalpha(0));
	return 0;
}
```

Devuelve:

return 0: Devuelve falso desde una función.
return 1: Devolviendo verdadero desde una función.

ft_toupper.c

Prototipo: int ft_toupper(int c)

c = El carácter a revisar.

Descripción: El objetivo de la función es comprobar si el carácter [c] está en minúscula y si es cierto, pasarlo a mayúscula.

int ft_toupper(int c) {	Prototipo de la función

if (c >= 'a' && c <= 'z')
{
c = c - 32;
}
	Si el carácter enviado [c] pertenece al entorno de las minúsculas.
	Entonces, conversión a mayúscula.

return (c); }	Devolución del carácter. Se mayúscula o minúscula.

Main:

```
int main()
{

	printf("Resultado: %c\n", ft_toupper('a'));
	return 0;
}
```

Devuelve:

“A”

ft_isdigit.c

Prototipo: int ft_isdigit(int c)

c = El carácter a revisar.

Descripción: El objetivo de la función es comprobar si el carácter [c] enviado es un dígito.

int ft_isdigit(int c) {	Prototipo de la función

if (c >= '0' && c <= '9')
{
return (1);
}
	Si el carácter enviado [c] se encuentra entre el 0 y el 9.
	Retorna 1 (End function)

return (0);	Si no, Retorna 0 (End function)

Main:

```
int main()
{

	printf("Resultado: %s\n", ft_isdigit(0));
	return 0;
}
```

Devuelve:

return 0: Devuelve falso desde una función.
return 1: Devolviendo verdadero desde una función.

ft_tolower.c

Prototipo: int ft_toupper(int c)

c = El carácter a revisar.

Descripción: El objetivo de la función es comprobar si el carácter [c] está en mayúscula y si es cierto, pasarlo a minúscula.

int ft_tolower(int c) {	Prototipo de la función

if (c >= 'A' && c <= 'Z')
{
c = c + 32;
}
	Si el carácter enviado [c] pertenece al entorno de las mayúsculas
	
	Entonces, conversión a minúscula

return (c); }	Devolución del carácter. Se mayúscula o minúscula.

Main:

```
int main()
{

	printf("Resultado: %c\n", ft_tolower('A'));
	return 0;
}
```

Devuelve:

“a”

ft_isalnum.c

Prototipo: int ft_isalnum(int c)

c = El carácter a revisar.

Descripción: El objetivo de la función es comprobar si el carácter [c] enviado es alfanumérico. Es equivalente a ft_isalpha.c y ft_isdigit.c

int ft_isalnum(int c) {	Prototipo de la función

if (c >= '0' && c <= '9')
{
return (1);
}
	Si el carácter enviado [c] se encuentra entre el '0' y el '9'.
	Retorna 1 (End function)

else if (c >= 'A' && c <= 'Z')
{
return (1);
}
	Si el carácter enviado [c] se encuentra entre la 'A' y la 'Z'.
	Retorna 1 (End function)

else if (c >= 'a' && c <= 'z')
{
return (1);
}
	Si el carácter enviado [c] se encuentra entre la 'a' y la 'z'.
	Retorna 1 (End function)

return (0); }	Si no, Retorna 0 (End function)

Main:

```
int main()
{

	printf("Resultado: %s\n", ft_isalnum(0));
	return 0;
}
```

Devuelve:

return 0: Devuelve falso desde una función.
return 1: Devolviendo verdadero desde una función.

ft_isascii.c

Prototipo: int ft_isalnum(int c)

c = El carácter a revisar.

Descripción: El objetivo de la función es comprobar si el carácter [c] enviado es un valor que encaje dentro del conjunto de caracteres ASCII.

int ft_isascii(int c) {	Prototipo de la función

if (c >= 0x00 && c <= 0x7f)
{
return (1);
}
	Sí el carácter [c] enviado se encuentra dentro de la tabla ASCII
	Retorna 1 (End fucntion).
	Representado en valores hexadeciamels

return (0); }	Si no lo es, retorna 0 (End function)

Main:

```
int main()
{

	printf("Resultado: %s\n", ft_isascii(0));
	return 0;
}
```

Devuelve:

return 0: Devuelve falso desde una función.
return 1: Devolviendo verdadero desde una función.

Tablas ASCII

Tabla 1: Caracteres ASCII de control

Dec	Hex	C	Car	Descripción	Description
0	0	\0	NUL	carácter nulo	null
1	1	\1	SOH	comienzo de cabecera	start of heading
2	2	\2	STX	comienzo de texto	start of text
3	3		ETX	fin de texto	end of text
4	4		EOT	fin de transmisión	end of transmission
5	5		ENQ	petición	enquiry
6	6		ACK	reconocimiento	acknowledge
7	7	\a	BEL	timbre	bell
8	8		BS	retroceso	backspace
9	9	\t	TAB	tabulador horizontal	horizontal tab
10	a	\n	LF/NL	salto de línea	line feed/new line
11	b	\v	VF	tabulador vertical	vertical tab
12	c	\f	FF/NP	salto de página	form feed/new page
13	d	\r	CR	retorno de carro	carriage return
14	e		SO	cambiar conjunto de caracteres	shift out
15	f		SI	volver al conjunto de caracteres	shift in
16	10		DLE	escape de enlace de datos	data link escape
17	11		DC1	control de dispositivo 1	device control 1
18	12		DC2	control de dispositivo 2	device control 2
19	13		DC3	control de dispositivo 3	device control 3
20	14		DC4	control de dispositivo 4	device control 4
21	15		NAK	reconocimiento negativo	negative acknowledge
22	16		SYN	espera síncrona	synchronous idle
23	17		ETB	fin de bloque de transmisión	end of transmission block
24	18		CAN	cancelar	cancel
25	19		EM	fin de medio	end of medium
26	1a		SUB	substitución	substitute
27	1b		ESC	escape	escape
28	1c		FS	separador de fichero	file separator
29	1d		GS	separador de grupo	group separator
30	1e		RS	separador de registro	record separator
31	1f		US	separador de unidad	unit separator
127	7f		DEL	suprimir	delete

Tabla 2: Caracteres ASCII imprimibles

Dec	Hex	Car	Dec	Hex	Car	Dec	Hex	Car	Dec	Hex	Car	Dec	Hex	Car	Dec	Hex	Car
32	20	espacio	48	30	0	64	40	@	80	50	P	96	60	`	112	70	p
33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
36	24	$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
39	27	'	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
40	28	(	56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
41	29	)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
42	2a	*	58	3a	:	74	4a	J	90	5a	Z	106	6a	j	122	7a	z
43	2b	+	59	3b	;	75	4b	K	91	5b	[	107	6b	k	123	7b	{
44	2c	,	60	3c	<	76	4c	L	92	5c	\	108	6c	l	124	7c	|
45	2d	-	61	3d	=	77	4d	M	93	5d	]	109	6d	m	125	7d	}
46	2e	.	62	3e	>	78	4e	N	94	5e	^	110	6e	n	126	7e	~
47	2f	/	63	3f	?	79	4f	O	95	5f	_	111	6f	o			

ft_isprint.c

Prototipo: int ft_isprint(int c)

c = El carácter a revisar.

Descripción: El objetivo de la función es comprobar si el carácter [c] enviado es un valor imprimible.

int ft_isprint(int c) {	Prototipo de la función

if (c >= 0x20 && c <= 0x7E)
{
return (1);
}
	Si el carácter enviado [c] pertenece a la parte imprimible de la tabla ASCII
	Retorna 1 (End function)

return (0); }	Si no, retorna 0 (End function)

Main:

```
int main()
{

	printf("Resultado: %s\n", ft_isprint(0));
	return 0;
}
```

Devuelve:

return 0: Devuelve falso desde una función.
return 1: Devolviendo verdadero desde una función.

ft_strlen.c

Prototipo: size_t ft_strlen(const char *s)

s = La cadena a revisar.

Descripción: El objetivo de la función es contar el largo de la cadena [s] enviada.

size_t ft_strlen(const char *s) {	Prototipo de la función
size_t i;	Variable para contar posiciones
i = 0;	Iniciamos la posición a ‘0’

while (s[i])
{
i++;
}
	Mientras la cadena enviada [s] no llegue a NUL
	
	Sumamos una posición a la variable [i]

return (i); }	Devolvemos el resultado total de las sumas realizadas con la variable [i]

Main:

```
int main()
{
	char s[] = “Esto es un test”;

	printf("Resultado: %s\n", ft_strlen(s));
	return 0;
}
```

Devuelve:

“15”

ft_strchr.c

Prototipo: char *ft_strchr(const char *str, int c)

str = La cadena a revisar.
c = carácter a buscar.

Descripción: El objetivo de la función es contar desde el principio de la cadena [str] el carácter similar al buscado [c]. Se devolverá el puntero una vez encontrado.

char *ft_strchr(const char *str, int c) {	Prototipo de la función
unsigned char ch;	Variable para pasar a char.
ch = c;	Pasamos el parámetro [c] a unsigned char

while (*str != '\0')
{
if (*str == ch)
{
return ((char *) str);
}
str++;
}
	Mientras la cadena [str] no llegue al final (NUL)
	
	Si, la posición de la cadena [str] es igual al carácter enviado [c].
	Devuelve la posición de la cadena [str] hacía delante. (End function)
	Si no, sumamos una posición a la cadena [str]

if (ch == '\0')
{
return ((char *) str);
}
	Si el carácter enviado [c] es igual a NUL
	
	Entonces devolvemos la posición de la cadena [str] hacía delante (End function)

return (NULL); }	Devolución de NUL si no se cumple ninguna condición (End function)

Main:

```
int main()
{
	Char text[] = “Me gusta programar en 42 Barcelona”;
	Char search = ‘g’;

	printf("Resultado: %s\n", ft_strchr(text, search));
	return 0;
}
```

Devuelve:

“gusta programar en 42 Barcelona”

ft_strrchr.c

Prototipo: char *ft_strrchr(const char *s, int c)

s = La cadena a revisar.
c = carácter a buscar.

Descripción: El objetivo de la función es contar desde el final de la cadnea [str] el carácter similar al buscado [c]. Se devolverá el puntero una vez encontrado.

char *ft_strrchr(const char *s, int c) {	Prototipo de la función
size_t len;	Variable para saber el largo de la cadena [s]
len = ft_strlen(s);	Iniciamos la variable con el largo total de la cadena

if ((char)c == '\0')
{
return ((char *)s + len);
}
	Sí, [c] convertida a -char- es igual a NUL (significa que está vacía)
	Entonces, devuelve la cadena [s] más el largo.

while (len > 0)
{
len--;
if (*(s + len) == (char) c)
{
return ((char *)s + len);
}
}
	Mientras, el largo sea mayor que '0'
	
	Restamos una posición a la cadena.
	Sí, la cadena [s] más el largo es igual que [c]
	
	Entonces, devuelve la cadena [s] más el largo.

return (NULL); }	Si no se cumple ninguna de las condiciones anteriores, devuelve NULL (End function)

Main:

```
int main()
{
	Char text[] = "Me gusta programar en 42 Barcelona";
	Char search = 'g';

	printf("Resultado: %s\n", ft_strchr(text, search));
	return 0;
}
```

Devuelve:

"gramar en 42 Barcelona"

ft_strncmp.c

Prototipo: int ft_strncmp(const char *s1, const char *s2, size_t n)

s1 = Cadena número 1 a revisar.
s2 = Cadena número 2 a revisar.
n = Número de caracteres a comparar.

Descripción: La función no compara más de n caracteres. Debido a que está diseñado para comparar cadenas en lugar de datos binarios, los caracteres que aparecen después de un carácter '\0' no se comparan.

La función devuelve un número entero mayor, igual o menos que 0, según la cadena s1 sea mayor, igual o menor que la cadena s2. La comparación se realiza utilizando caracteres sin signo.

`int ft_strncmp(const char *s1, const char *s2, size_t n)` `{`	Prototipo de la función
`size_t i;`	Variable para contar posiciones

`if (n == 0)`
`{`
`return (0);`
`}`
	Sí, el parámetro [n] está a cero.
	
	Entonces, devolveremos ‘0’

`i = 0;`	Iniciamos la posición a ‘0’

`while (i < n)`
`{`
`if (s1[i] != '\0' && s1[i] == s2[i])`
`{`
	Mientras, que la posición [i] sea menos al número de número de caracteres [n]
	Sí, la posición [s1] no es NUL y la posición de la cadena [s1] es igual a la posición de la cadena [s2]

`while (s1[i] != '\0' && s1[i] == s2[i] && i < n)`
`i++;`
`}`
	Mientras, la posición [s1] no es NUL y la posición [s1] es igual a la posición [s2] y la posición es menor al número de caracteres buscado [n]
	Sumaremos una posición a la variable [i]

`else`
`return ((unsigned char)s1[i] - (unsigned char)s2[i]);`
`}`
	Entonces, si no se cumple ninguna condición.
	Devolvemos la resta entre la posición de la cadena [s1] menos la posición de la cadena [s2]. (resta entre los números ASCII)

`return (0);` `}`	Si no se cumple ninguna condición, devolvemos 0. (End function)

Main:

```
int main()
{
	char s1[] = "Me gusta programar en 42 Barcelona";
	char s2[] = "Me gusta Programar en 42 Barcelona";
	unsigned int n = 15;

	printf("Resultado %d\n", ft_strncmp(s1, s2, n));
	return 0;
}
```

Devuelve:

“32”

ft_bzero.c

Prototipo: void ft_bzero(void *s, size_t n)

s = Cadena a enviar.
n = Número de caracteres a tratar.

Descripción: La función es poner a 0 los [n] primero caracteres de la cadena [s]. Sí [n] es igual a ‘0’ la función no realizará ninguna acción.

void ft_bzero(void *s, size_t n) {	Prototipo de la función
size_t i; char *src;	Variable para contar posiciones. Variable para convertir la cadena a char
i = 0; src = s;	Iniciamos la posición a ‘0’ Igualamos la cadena a char [src]

while (n > 0)
{
src[i] = '\0';
i++;
n--;
}
}
	Mientras, el número de caracteres a tratar [n] sea mayor a ‘0’
	La posición [i] de la cadena [src] es igual a NUL
	Sumamos una posición.
	Restamos el número de caracteres a tratar [n]

Main:

```
int main()
{
	char s[] = "Me gusta programar en 42 Barcelona";

	ft_bzero(s, 10);
	return 0;
}
```

Devuelve:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
\0	\0	\0	\0	\0	s	t	a		p	r	o	g	r	a	m	a	r	\0

ft_memset.c

Prototipo: void *ft_memset(void *b, int c, size_t len)

b = Cadena a enviar.
c = Carácter a copiar.
len = cantidad a repetir

Descripción: Copia el carácter [c] (un char sin signo) a los primeros [len] caracteres de la cadena [b]

void *ft_memset(void *b, int c, size_t len) {	Prototipo de la función
unsigned char *src; size_t i;	Variable para convertir la cadena a char Variable para sumar posiciones
src = (unsigned char *)b; i = 0;	Igualamos la cadena a char [b] Iniciamos la posición a '0'

while (i < len)
{
*(src + i) = (unsigned char)c;
i++;
}
	Mientras, la posición [i] sea menos que la cantidad a repetir [len]
	La posición de la cadena será igual al carácter a copiar.
	Sumaremos una posición [i]

return (b); }	Retornaremos la cadena envaida.

Main:

```
int main()
{
	char str[35];

	strcpy(str,"This is string.h library function");
	puts(str);
	ft_memset(str,'g',7);
	puts(str);
	return(0);
}
```

Devuelve:

"Me gusta programar en 42 Barcelona"
"gggggggga programar en 42 Barcelona"

ft_memchr.c

Prototipo: void *ft_memchr(const void *s, int c, size_t n)

s = Cadena a enviar.
c = Carácter a buscar.
n = Cantidad posiciones a buscar.

Descripción: La función localiza la primera aparición de c (convertido en char sin signo) en la cadena [s]. Devuelve un puntero al byte ubicado, o NUL si no existe tal byte dentro de n bytes.

void *ft_memchr(const void *s, int c, size_t n) {	Prototipo de la función
unsigned char *src; size_t i;	Variable para convertir la cadena [s] en char Variable para contar posiciones
src = (unsigned char *)s; i = 0;	Igualamos la cadena a char [c] Iniciamos la posición a '0'

while (i != n)
{
if (src[i] == ((unsigned char)c))
{
return (src + i);
}
i++;
}
	Mientras, la posición [i] no sea igual a la cantidad de posiciones a buscar [n]
	Sí, la posición de la cadena es igual al valor buscado [c] (Convertido en char)
	Entonces, devolverá la cadena desde la posición [i]
	
	Sumar una posición.

return (NULL); }	Si ninguna de las condiciones se cumple, retorna NULL (End function)

Main:

```
int main()
{
	char s[] = "Me gusta programar en 42 Barcelona";

	printf("Cadena: %s\n", s);
	printf("Resultado %s\n", ft_memchr(s,'4',35));
	return(0);
}
```

Devuelve:

"Me gusta programar en 42 Barcelona"
"42 Barcelona"

ft_memcpy.c

Prototipo: void *ft_memcpy(void *dst, const void *src, size_t n)

dst = Cadena dst donde se copiará la cadena src.
src = Cadena src a copiar en dst.
n = Cantidad de veces hasta dejar de copiar.

Descripción: La función copia [n] caracteres del área de la memoria de [src] en el área de la memoria de [dst].

void *ft_memcpy(void *dst, const void *src, size_t n) {	Prototipo de la función
char *sorc; char *dest; size_t i;	Variable para convertir la cadena [src] Variable para convertir la cadena [dst] Variable para contar posiciones

sorc = (char *)src;
dest = dst;
i = 0;
	Igualamos la cadena [src] a char (conversión de const a char)
	Igualamos la cadena a char [dst] a char
	Iniciamos la posición a '0'

if (sorc == NULL && dest == NULL)
{
return (NULL);
}
	Si la cadena [src] está vacía y la cadena [dst] está vacía
	Entonces, retornaremos NULL (End function)

while (i < n)
{
dest[i] = sorc[i];
i++;
}
	Mientras, la posición [i] sea menos que la cantidad de veces a copiar [n]
	La cadena destino [dst] será igual que la cadena a enviar [src]
	Sumamos una posición.

return (dest); }	Devolvemos la cadena destino [dst]

Main:

```
int main()
{
	char src[] = "Me gusta programar en 42 Barcelona";
	char dst[] = "42 Barcelona";

	printf("Resultado %s\n", ft_memcpy(dst, src, 9));
	return(0);
}
```

Devuelve:

"Me gusta programar en 42 Barcelona"
"Me gusta ona"

ft_memcmp.c

Prototipo: int ft_memcmp(const void *s1, const void *s2, size_t n)

s1 = Cadena a enviar y comparar.
s2 = Cadena a enviar y comparar.
n = Cantidad de bytes a comparar.

Descripción: La función compara los primeros [n] bytes de las áreas de memoria [s1] y [s2]. D

Devuelve ‘0’ si las dos cadenas son idénticas, de lo contrario, devuelve un entero menor, igual o mayor que cero si [s1] es, respectiva- mente, menor, igual o mayor que [s2]. (tratados como valores char sin signo).

int ft_memcmp(const void *s1, const void *s2, size_t n) {	Prototipo de la función
unsigned char *str1; unsigned char *str2; size_t i;	Variable para convertir cadena a unsigned char Variable para convertir cadena a unsigned char Variable para contar posiciones

str1 = (unsigned char *)s1;
str2 = (unsigned char *)s2;
i = 0;
	Igualamos la cadena [s1]
	Igualamos la cadena [s2]
	Iniciamos posición a ‘0’

if (s1 == s2)
{
return (0);
}
	Sí, la cadena [s1] es igual a la cadena [s2]
	
	Retornamos ‘0’ (End function)

while (i < n)
{
if (str1[i] != str2[i])
{
return (str1[i] - str2[i]);
}
i++;
}
	Mientras, que la posición [i] sea menor que la cantidad de bytes a comparar [n]
	Sí, la cadena [s1] no es igual a la cadena [s2]
	
	Retornaremos la resta de ambas cadenas [s1] y [s2]. (End function)
	Sumamos una posición [i]

return (0); }	Si ninguna condición de cumple, retornaremos ‘0’ (End function)

Main:

```
int main()
{
	char s1[] = "Me gusta programar en 42 Barcelona";
	char s2[] = "Me gusta programar en 24 Barcelona";

	printf("Resultado %d\n", ft_memcmp(s1, s2, 35));
	return 0;
}
```

Devuelve:

“2”

ft_memmove.c

Prototipo: void *ft_memmove(void *dst, const void *src, size_t len)

dst = Cadena donde se copiará el contenido.
src = Cadena se copiará.
len = Número de bytes que se copiarán.

Descripción: El objetivo de la función es copiar una cantidad determinada de [n] caracteres desde la cadena [src] a la cadena [dst].

static void ft_strcpy(void *dst, const void *src, size_t len) {	Prototipo auxiliar para hacer el copy
char *dest; char *sorc; size_t i;	Variable para pasar a char la cadena[dst] Variable para pasar a char la cadena[src] Variable para contar posiciones

dest = (char *)dst;
sorc = (char *)src;
i = 0;
	Convertimos la cadena [dst] a char [dest]
	Convertimos la cadena [src] a char [sorc]
	Iniciamos la posición a '0'

if (dest < sorc)
{
while (i < len)
{
dest[i] = sorc[i];
i++;
}
}
	Sí, la cadena [dest] es menor que la cadena [sorc]
	
	Mientras, la posición [i] sea menor a la cantidad de caracteres a copiar [len]
	La posición de la cadena [dest] será igual a la posición de la cadena [sorc]
	Sumamos una posición

if (dest > sorc)
{
while (len > 0)
{
dest[len - 1] = sorc[len - 1];
len--;
}
}
}
	Sí, la cadena [dest] es mayor que la cadena [sorc]
	
	Mientras que la cantidad de caracteres a copiar [len] sea mayor que '0'
	La posición [dest](total a copiar menos 1) será igual a la posición [sorc](total a copiar menos 1).
	Restamos un carácter a copiar [len]

void *ft_memmove(void *dst, const void *src, size_t len) {	Prototipo de la función

if (dst == src || !len)
{
return (dst);
}
	Sí, la cadena [dst] es igual a la cadena [src] o la cantidad a copiar es NULO (no se ha informado)
	Retornaremos la cadena entera de [dst]
	(End function)

ft_strcpy(dst, src, len);	Si no se cumple la regla, pasaremos las cadenas y la cantidad en la función auxiliar.
return (dst); }	Retornamos la cadena entera de [dst] (End function)

Main:

```
int main()
{
	char src[] = "42 Barcelona";
	char dst[] = "Me gusta programar en 42 Barcelona";

	printf("Source: %s\n", src);
	printf("Destiny: %s\n", dst);
	printf("Resultado %s\n", ft_memmove(dst, src, 12));
	return 0;
}
```

Devuelve:

"Source: 42 Barcelona"
"Destinty: Me gusta programar en 42 Barcelona"
"Resultado: 42Barcelongramar en 42 Barcelona"

ft_strlcpy.c

Prototipo: size_t ft_strlcpy(char *dst, const char *src, size_t size)

dst = Cadena a enviar para ser copiada desde [src].
src = Cadena a enviar para copiar en [dst].
size = Cantidad de bytes a copiar.

Descripción: La función copia la cadena [src] hasta el tamaño[size] -1 caracteres a la cadena [dst]. El -1 es para poder añadir NUL al terminar. Devuelve el tamaño de [src]

size_t ft_strlcpy(char *dst, const char *src, size_t size) {	Prototipo de la función
size_t i; size_t cnt;	Variable para contar posiciones Variable para contar el largo de la cadena [src]
i = 0; cnt = ft_strlen(src);	Iniciamos la posición a '0' Contamos el largo de la cadena [src]

while (src[i] && i + 1 < size)
{
dst[i] = src[i];
i++;
}
	Mientras, la posición de [src] y la posición [i] más 1 sea menor a la cantidad de bytes a copiar
	La posición de [dst] será igual a la posición de [src]
	Sumamos una posición

if (size != '\0') dst[i] = '\0';	Sí, La cantidad a copiar no es igual a NUL La posición de [dst] será igual a NUL
return (cnt); }	Devolveremos el largo de la cadena [src] (End function)

Main:

```
int main()
{
	char src[] = "42 Barcelona";
	char dst[] = "Me gusta programar";

	printf("src antes: %s\n", src);
	printf("dst antes: %s\n", dst);
	printf("Resultado %d\n", ft_strlcpy(dst, src, 3));
	printf("src después: %s\n", src);
	printf("dst después: %s\n", dst);
	return 0;
}
```

Devuelve:

"src antes: 42 Barcelona"
"dst antes: Me gusta programar"
"Resultado 12"
"src después: 42 Barcelona"
"dst después: 42"

ft_strlcat.c

Prototipo: size_t ft_strlcat(char *dst, const char *src, size_t dstsize)

dst = Cadena a enviar.
src = Cadena a enviar.
dstsize = Longitud de la cadena.

Descripción: La función agregar una cadena al final de la otra, estableciendo la longitud de la cadena de destino.

size_t ft_strlcat(char *dst, const char *src, size_t dstsize) {	Prototipo de la función
size_t i; size_t j; size_t res_a; size_t res_b;	Variable para la posición final de la cadena [dst] Variable para contar posiciones Variable para contar largo de [dst] Variable para contar largo de [src]

i = ft_strlen(dst);
j = 0;
res_a = ft_strlen(dst);
res_b = ft_strlen(src);
	Iniciamos variable con el largo de la cadena [dst]
	Iniciamos variable en la posición '0'
	Iniciamos variable con el largo de la cadena [dst]
	Iniciamos variable con el largo de la cadena [src]

if (dstsize < 1) return (res_b + dstsize);	Sí, la longitud de la cadena es menor a 1 Retornamos el total de la cadena [src] más la longitud de la cadena.

while (src[j] != '\0' && i < dstsize - 1)
{
dst[i] = src[j];
i++;
j++;
}
	Mientras, la posición [j] de la cadena [src] no llegue a NULL y el largo de la cadena [dst] sea menor que la longitud de la cadena menos 1
	La posición [i] de la cadena [dst] será igual a la posición [j] de la cadena [src]
	Sumamos una posición [i] al largo de la cadena [dst]
	Sumamos una posición [j] a la cadena [src]

dst[i] = '\0';	Añadimos NUL al final de la posición [i] de la cadena [dst]
if (dstsize < res_a) return (res_b + dstsize);	Sí, la longitud de la cadena [dstsize] es menor que el total de la cadena [dst] Retornamos la suma del total de la cadena [src] más la longitud de la cadena

else
return (res_a + res_b);
}
	Si no,
	Retornamos la suma de la cadena [dst] más [src]

Main:

```
int main()
{
     char src[] = "42 Barcelona";
     char dst[] = "Me gusta programar";

     printf("src antes: %s\n", src);
     printf("dst antes: %s\n", dst);
     printf("Resultado %d\n", ft_strlcat(dst, src, sizeof(dst)));
     printf("src después: %s\n", src);
     printf("dst después: %s\n", dst);
     return 0;
}
```

Devuelve:

"src antes: 42 Barcelona"
"dst antes: Me gusta programar en"
"Resultado 34"
"src después: 42 Barcelona"
"dst después: Me gusta programar en"

ft_strnstr.c

Prototipo: char *ft_strnstr(const char *haystack, const char *needle, size_t len)

haystack = Cadena a revisar.
needle = Cadena que debe coincidir o no con [haystack]
len = Longitud de caracteres a revisar.

Descripción: La función localiza la primera aparición de la cadena terminada en nulo [needle] en la cadena [haystack], donde no se buscan más de [len] caracteres.

Si [needle] es una cadena vacía, se devuelve [haystack]. Si [needle] no aparece en ninguna parte de [haystack], se devuelve NUL. De lo contrario, se devolverá un puntero al primer carácter de la primera aparición de [needle].

size_t len_x(const char *haystack, const char *needle, size_t len, size_t i) {	Prototipo auxiliar para saber el largo de [x]
size_t x;	Variable para contar posiciones
x = 0;	Iniciamos la posición a '0'

while (haystack[i + x] != '\0' && needle[x] != '\0'
&& haystack[i + x] == needle[x] && i + x < len)
{
x++;
}
	Mientras que la posición de la cadena a revisar no sea NUL **y** la posición de la cadena que debe coincidir no sea NUL **y** mientras que la posición de la cadena a revisar no sea igual que la posición de la cadena a coincidir **y** que ambas posiciones sean menores que la longitud a revisar
	Sumamos una posición a [x]

return (x); }	Retorna la suma total de posiciones [x]
char *ft_strnstr(const char *haystack, const char *needle, size_t len) {	Prototipo de la función
size_t i; size_t x; size_t count;	Variable para la posición. Variable para sumar posiciones. Variable para el largo de la cadena a coincidir [needle]
i = 0; count = ft_strlen(needle);	Iniciamos la posición a '0' Calculamos el total de bytes de la cadena a coincidir.

if (needle[0] == '\0' || haystack == needle)
{
return ((char *) haystack);
}
	Sí, la cadena a coincidir no es igual a NUL **o** la cadena a revisar es igual a la cadena a coincidir
	Retornamos la cadena a revisar

while (haystack[i] != '\0' && i < len)
{
x = len_x(haystack, needle, len, i);
if (x == count)
{
return ((char *) haystack + i);
}
i++;
}
	Mientras que la posición de la cadena a revisar no llegue a NUL **y** la posición [i] sea menor que la longitud de caracteres a revisar.
	Contamos la cantidad de posiciones [x]
	Sí, la cantidad de posiciones [x] es igual al largo de la cadena a coincidir
	Retornamos la cadena a revisar desde la posición [i]
	Si no se cumple la condición sumamos una posición a [i]

return (NULL); }	Si ninguna condición se cumple retornamos NULL (End function)

Main:

```
int main()
{
	char haystack[] = "Me gusta programar en 42 Barcelona";
	char needle[] = "42 Barcelona";

	printf("%s",ft_strnstr(haystack, needle, 35));
	return (0);
}
```

Devuelve:

"42 Barcelona"

ft_atoi.c

Prototipo: int ft_atoi(const char *str)

str = Cadena a convertir en número

Descripción: El objetivo de la función es convertir una cadena de números char a números integer.

int ft_space(char c) {	Función auxiliar para comprobar tabulaciones, espacios, etc…

return (c == '\t' || c == '\n' || c == '\v' || \
c == '\f' || c == '\r' || c == ' ');
}
	Retornamos si [c] coincide con algunos de estos caracteres no imprimibles (Ver cuadro)

int ft_atoi(const char *str) {	Prototipo de la función
int res; int sign; int i;	Variable para almacenar el resultado Variable para almacenar el signo Variable para contar posiciones

res = 0;
sign = 1;
i = 0;
	Iniciamos el resultado a ‘0’
	Iniciamos el signo en positivo
	Iniciamos la posición a ‘0’

if (str[i] == '\0') return (0);	Sí, la posición de [str] es igual a NUL Retornamos ‘0’
while (ft_space(str[i])) i++;	Mientras, que la posición de la cadena sea un carácter no imprimible Sumamos una posición a la [i]

if (str[i] == '-')
{
sign = -1;
i++;
}
	Si, la posición de la cadena es igual a ‘menos’
	
	Multiplicamos para pasar el valor del signo a ‘-1’
	Sumamos una posición a la [i]

else if (str[i] == '+') i++;	Entonces sí, la posición de la cadena es igual a ‘más’ Sumamos una posición a la [i]

while (str[i] >= '0' && str[i] <= '9')
{
res = res * 10 + str[i] - '0';
i++;
}
	Mientras, la posición de la cadena sea mayor a ‘0’ y la posición de la cadena sea menor que ‘9’
	El resultado de multiplica por 10 en la posición de la cadena y se resta ‘0’ para convertirlo a -int-.
	Sumamos una posición a la [i]

return (res * sign); }	Retornamos el resultamos multiplicado por el signo.

Main:

```
int main()
{
	char str[] = "  -1235645665";

	printf("%d\n",ft_atoi(str));
	return (0);
}
```

Devuelve:

“-1235645665”

ft_calloc.c

Prototipo: void *ft_calloc(size_t count, size_t size)

count = Matriz de elemtos
size = Total de bytes

Descripción: Asigna memoria para una matriz de elementos de bytes cada uno y devuelve un puntero a la memoria asignada. La memoria es puesta a cero.

void *ft_calloc(size_t count, size_t size) {	Prototipo de la función
void *ptr;	Variable para la memoria (Void porque no sabemos el valor que se pondrá dentro)

ptr = (void *)malloc(sizeof(void) * (count * size));
if (!ptr)
{
return (NULL);
}
	Reservamos un espacio de memoria en el Heap.
	Sí, no tenemos espacio
	
	Retornamos NULL

else
{
ft_bzero(ptr, count * size);
return (ptr);
}
}
	Entonces,
	
	Ponemos cada byte a cero.
	Retornamos el puntero de la memoria reservada en el Heap

Main:

```
int main()
{
	printf("%p\n",ft_calloc(5, 2));
	return (0);
}
```

Devuelve:

“0x5402040”

ft_strdup.c

Prototipo: char *ft_strdup(const char *s1)

s1 = Cadena a duplicar

Descripción: La función devuelve un puntero a una nueva cadena que es un duplicado de la cadena [s1]. La memoria para la nueva cadena es obtenida con malloc(3).

char *ft_strdup(const char *s1) {	Prototipo de la función
char *ptr; size_t len;	Variable para la reserva de memoria Variable para saber el largo de la cadena
len = ft_strlen(s1) + 1;	Iniciamos contando el largo de la cadena [s1]

ptr = malloc(sizeof(char) * (len));
if (!ptr)
{
return (NULL);
}
	Reservamos memoria en el Heap
	Si, no hay memoria
	
	Retornamos NULL (End function)

else
{
ft_strlcpy((char *)ptr, s1, len);
return (ptr);
}
}
	Entonces,
	
	Copiamos la cadena
	Devolvemos el puntero de la reserva de memoria

Main:

```
int main()
{
	char s1[] = "Me gusta programar en 42 Barcelona";

	printf("%s\n",ft_strdup(s1));
	return (0);
}
```

Devuelve:

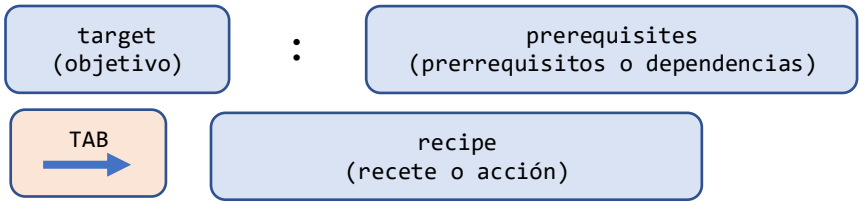
“Me gusta programar en 42 Barcelona”

Makefile

Descripción: Es una herramienta para simplificar u organizar el código para la compilación con un conjunto de comandos (similares a los comandos de terminal) con nombres de variables y objetivos para crear archivos de objetos y eliminarlos.

En un solo archivo de creación, podemos crear múltiples objetivos para compilar y eliminar objetos.

Puede compilar su proyecto (programa) cualquier número de veces usando Makefile.



Target:
1. Archivo generado por el programa
2. Nombre o acción

Prerrequisitos:
Archivo que es usado para generar el target
➔ Si un prerrequisito cambia se ejecutra la receta (acción)

NAME = libft.a HEADER = libft.h	Nombre del programa
CC = gcc FLAGS = -Wall -Werror -Wextra AR = ar -rcs RM = rm -f	Variables
SRCS = ft_strlcat.c ft_strlen.c ft_strncmp.c \ ft_toupper.c ft_tolower.c ft_isalpha.c \ ft_isdigit.c ft_isalnum.c ft_isascii.c \ ft_isprint.c ft_strlcpy.c ft_strchr.c \ ft_strrchr.c ft_strnstr.c ft_memchr.c \ ft_memset.c ft_bzero.c ft_memcpy.c \ ft_memmove.c ft_memcmp.c ft_atoi.c \ ft_calloc.c ft_strdup.c ft_striteri.c \ ft_strmapi.c ft_putchar_fd.c ft_putstr_fd.c \ ft_putendl_fd.c ft_putnbr_fd.c ft_substr.c \ ft_strjoin.c ft_strtrim.c ft_split.c ft_itoa.c	Nombre de los archivos
OBJS = $(SRCS:.c=.o)	Remplazar los archivos por .c a .o
%.o: %.c $(HEADER) $(CC) $(FLAGS) -I./ -c $< -o $@	Generará un archivo .o a partir de un archivo .c
all: $(NAME)	Se compila el archivo binario (ejecutable)
$(NAME): ${OBJS} ${HEADER} $(AR) $(NAME) $(OBJS)	Se compila los objetos con las librerías y archivos
clean: $(RM) $(OBJS)	Eliminar todos los archivos .o
fclean: clean $(RM) $(NAME)	Elimina todos los archivos .o y binarios (ejecutable)
re: fclean all	Hace un re-make, es como si se hubiera ejecutado Make por primera vez
.PHONY: all clean fclean re	Le dice al make que estos nombres no son archivos

Notas:

ar = crea, modifica y extrae archivos. Un archivo es un único archivo que contiene una colección de otros archivos en una estructura que hace posible recuperar los archivos individuales originales

-r = (agregar con reemplazar) para agregar los mismos al archivo de la biblioteca
-c = (crear) para crear el archivo de la biblioteca
-s = (indexar) para crear un índice de los archivos dentro de la biblioteca.

rm = Eliminar archivos
-f = No solicita confirmación por lo que lo elimina todo sin pedir permisos.

$< = Sólo coge el primero de las dependencias (input)
$@ = Se remplaza por el nombre del objetivo. Es decir, que automáticamente pasaría de .c a .o o lo que ponga en objetivo [OBJS] es lo que cogerá (output)