Lista de Funciones en C

joaquín

Diciembre 2024

Funciones para Manejo de Caracteres

- **isalpha**: Verifica si un carácter es alfabético (letra).
  `int ft_isalpha(int c);`
  Parámetros: `c` - Carácter a evaluar.
  Retorna: Un valor distinto de cero si es una letra, 0 en caso contrario.

- **isdigit**: Verifica si un carácter es un dígito decimal ('0'-'9').
  `int ft_isdigit(int c);`
  Parámetros: `c` - Carácter a evaluar.
  Retorna: Un valor distinto de cero si es un dígito, 0 en caso contrario.

- **isalnum**: Verifica si un carácter es alfanumérico (letra o dígito).
  `int ft_isalnum(int c);`
  Parámetros: `c` - Carácter a evaluar.
  Retorna: Un valor distinto de cero si es alfanumérico, 0 en caso contrario.

- **isascii**: Verifica si un carácter pertenece al conjunto de caracteres ASCII.
  `int ft_isascii(int c);`
  Parámetros: `c` - Carácter a evaluar.
  Retorna: Un valor distinto de cero si es un carácter ASCII, 0 en caso contrario.

- **isprint**: Verifica si un carácter es imprimible.
  `int ft_isprint(int c);`
  Parámetros: `c` - Carácter a evaluar.
  Retorna: Un valor distinto de cero si es imprimible, 0 en caso contrario.

- **toupper**: Convierte un carácter en mayúsculas.
  `int ft_toupper(int c);`
  Parámetros: `c` - Carácter a convertir.
  Retorna: El carácter convertido en mayúsculas si es una letra minúscula, de lo contrario retorna el carácter sin cambios.

- **tolower**: Convierte un carácter en minúsculas.
  `int ft_tolower(int c);`
  Parámetros: `c` - Carácter a convertir.
  Retorna: El carácter convertido en minúsculas si es una letra mayúscula, de lo contrario retorna el carácter sin cambios.

Funciones de Cadenas

- **strlen**: Calcula la longitud de una cadena.
  `size_t ft_strlen(const char *s);`
  Parámetros: `s` - Cadena de la que calcular la longitud.
  Retorna: La longitud de la cadena.

- **strchr**: Busca la primera ocurrencia de un carácter en una cadena.
  `char *ft_strchr(const char *s, int c);`
  Parámetros: `s` - Cadena en la que buscar, `c` - Carácter a buscar.
  Retorna: El puntero a la primera ocurrencia de `c` o `NULL` si no se encuentra.

- **strrchr** : Busca la última ocurrencia de un carácter en una cadena.
`char *ft_strchr(const char *s, int c);`
Parámetros: `s` - Cadena en la que buscar, `c` - Carácter a buscar.
Retorna: El puntero a la última ocurrencia de `c` o `NULL` si no se encuentra.

- **strlcpy** : Copia una cadena a un búfer, garantizando seguridad.
`size_t ft_strlcpy(char *dst, const char *src, size_t dstsize);`
Parámetros: `dst` - Búfer de destino, `src` - Cadena de origen, `dstsize` - Tamaño del búfer de destino.
Retorna: La longitud de la cadena de origen (`src`). Si esta longitud es mayor o igual a `dstsize`, la cadena es truncada en el destino, garantizando la terminación nula.

- **strlcat** : Concatena cadenas de manera segura en un búfer limitado.
`size_t ft_strlcat(char *dst, const char *src, size_t dstsize);`
Parámetros: `dst` - Búfer de destino que contiene la cadena inicial, `src` - Cadena a concatenar al final de `dst`, `dstsize` - Tamaño total del búfer de destino (incluye el espacio para el terminador nulo). **Retorna:** La longitud que tendría la cadena concatenada si no hubiera truncamiento. Si el resultado es mayor o igual a `dstsize`, la concatenación fue truncada.
Descripción: Esta función agrega la cadena `src` al final de `dst`, asegurando que el resultado sea una cadena válida terminada en `\0` y evitando desbordamientos del búfer. **Nota:** Si `dstsize` es menor o igual a la longitud inicial de `dst`, no se realiza concatenación y se retorna el tamaño requerido.

- **strncmp** : Compara las primeras `n` caracteres de dos cadenas.
`int ft_strncmp(const char *s1, const char *s2, size_t n);`
Parámetros: `s1` - Primera cadena, `s2` - Segunda cadena, `n` - Número máximo de caracteres a comparar.
Retorna: Un valor menor que 0 si `s1` es menor que `s2`, 0 si las primeras `n` posiciones de `s1` y `s2` son iguales, Un valor mayor que 0 si `s1` es mayor que `s2`. La comparación se realiza hasta que se alcance el número máximo de caracteres especificado o hasta encontrar un carácter nulo (`\0`).

- **strnstr** : Busca la primera ocurrencia de una subcadena dentro de una cadena, limitando la búsqueda a los primeros `n` caracteres. Si `little` es una cadena vacía, la función devuelve `big`. La búsqueda se realiza solo hasta el número de caracteres especificado por `len`.
`char *ft_strnstr(const char *big, const char *little, size_t len);`
Parámetros: `big` - Cadena en la que buscar, `little` - Subcadena a buscar, `len` - Número máximo de caracteres a examinar.
Retorna: Un puntero a la primera ocurrencia de `little` en `big` dentro de los primeros `len` caracteres o `NULL` si no se encuentra la subcadena o si el número de caracteres examinados es insuficiente.

Funciones de Manipulación de Cadenas

- **substr** : Extrae una subcadena de la cadena original.
`char *ft_substr(char const *s, unsigned int start, size_t len);`
Parámetros: `s` - Cadena original, `start` - Índice de inicio, `len` - Longitud de la subcadena.
Retorna: La subcadena extraída, o `NULL` si hay un error de memoria.

- **strjoin** : Concatena dos cadenas.
`char *ft_strjoin(char const *s1, char const *s2);`
Parámetros: `s1` - Primera cadena, `s2` - Segunda cadena.
Retorna: La cadena concatenada, o `NULL` si hay un error de memoria.

- **strtrim** : Elimina todos los caracteres de una cadena desde el principio y final.
`char *ft_strtrim(char const *s1, char const *set);`
Parámetros: `s1` - Cadena a recortar, `set` - Caracteres a eliminar.
Retorna: La cadena recortada, o `NULL` si hay un error de memoria.
///HOLA// → HOLA (quita /)

- **split** : Separa una cadena en un array de cadenas.
`char **ft_split(char const *s, char c);`
Parámetros: `s` - Cadena a separar, `c` - Carácter delimitador.
Retorna: El array de cadenas resultante, o `NULL` si hay un error de memoria.

- **itoa** : Convierte un número entero en su representación en cadena.
`char *ft_itoa(int n);`

Parámetros: `n` - Número a convertir.
Retorna: La representación en cadena del número, o `NULL` si hay un error de memoria.

- **strmapi**: Aplica una función a cada carácter de la cadena.
`char *ft_strmapi(char const *s, char (*f)(unsigned int, char));`
Parámetros: `s` - Cadena a modificar, `f` - Función a aplicar a cada carácter.
Retorna: La nueva cadena resultante, o `NULL` si hay un error de memoria.
Ejemplo de funcion f, dentro del código se llamaría: `str[i] = f(i, s[i]);`

```
char f(unsigned int i, char c)
{
    return (c + i);
}
```

- **striteri**: Aplica una función a cada carácter de la cadena (modificando la cadena).
`void ft_striteri(char *s, void (*f)(unsigned int, char *));`
Parámetros: `s` - Cadena a modificar, `f` - Función a aplicar.
Retorna: Nada. Modifica la cadena en el lugar.
Ejemplo de funcion f, dentro del código se llamaría: `f(i, &s[i]);`

```
void        f(unsigned int i, char *c)
{
    c[i] = 'H';
}
```

Funciones de Memoria

- **memset**: Inicializa un bloque de memoria con un valor determinado.
`void *ft_memset(void *s, int c, size_t n);`
Parámetros: `s` - Bloque de memoria, `c` - Valor de inicialización, `n` - Número de bytes a modificar.
Retorna: El bloque de memoria modificado.

- **bzero**: Pone a cero un bloque de memoria.
`void *ft_bzero(void *s, size_t n);`
Parámetros: `s` - Bloque de memoria, `n` - Número de bytes a poner a cero.
Retorna: El bloque de memoria modificado.

- **memcpy**: Copia un bloque de memoria a otro.
`void *ft_memcpy(void *dest, const void *src, size_t n);`
Parámetros: `dest` - Destino, `src` - Origen, `n` - Número de bytes a copiar.
Retorna: El bloque de memoria de destino.

- **memmove**: Mueve un bloque de memoria a otro, manejando solapamientos.
`void *ft_memmove(void *dest, const void *src, size_t n);`
Parámetros: `dest` - Destino, `src` - Origen, `n` - Número de bytes a mover.
Retorna: El bloque de memoria de destino.
Copia de derecha a izquierda o al contrario según se situen los bloques

- **memchr**: Busca un valor en un bloque de memoria.
`void *ft_memchr(const void *s, int c, size_t n);`
Parámetros: `s` - Bloque de memoria, `c` - Valor a buscar, `n` - Número de bytes a buscar.
Retorna: El puntero a la primera ocurrencia de `c` o `NULL` si no se encuentra.

- **memcmp**: Compara un bloque de memoria con otro. La comparación se realiza byte por byte durante los primeros `n` bytes. Si los bloques de memoria son idénticos, devuelve 0.
`int ft_memcmp(const void *s1, const void *s2, size_t n);`
Parámetros: `s1` - Primer bloque de memoria, `s2` - Segundo bloque de memoria, `n` - Número de bytes a comparar.
Retorna:
  - Un valor menor que 0 si el bloque de memoria `s1` es menor que `s2`.
  - 0 si los bloques de memoria son idénticos.
  - Un valor mayor que 0 si el bloque de memoria `s1` es mayor que `s2`.

- **calloc**: Asigna memoria dinámica para un número de elementos y los inicializa a cero.
`void *ft_calloc(size_t count, size_t size);`
Parámetros: `count` - Número de elementos a asignar, `size` - Tamaño de cada elemento.
Retorna: Un puntero a la memoria asignada, o `NULL` si no se pudo realizar la asignación.

- **strdup**: Duplica una cadena de caracteres.
`char *ft_strdup(const char *s);`
Parámetros: `s` - Cadena que se va a duplicar.
Retorna: Un puntero a una nueva cadena que contiene una copia de `s`, o `NULL` si no se pudo asignar memoria.
Nota: La memoria asignada para la nueva cadena debe ser liberada después de su uso con `free`.

Funciones de Conversión

- **atoi**: Convierte una cadena a un número entero.
`int ft_atoi(const char *nptr);`
Parámetros: `nptr` - Cadena a convertir.
Retorna: El número entero representado por la cadena.

Funciones de Entrada/Salida

- **ft_putchar_fd**: Envia un carácter al file descriptor especificado.
`void ft_putchar_fd(char c, int fd);`
Parámetros: `c`- El carácter a enviar, `fd`- El file descriptor sobre el que escribir.
Retorna: Nada.

- **ft_putstr_fd**: Envía una cadena al file descriptor especificado.
`void ft_putstr_fd(char *s, int fd);`
Parámetros: `s`- La cadena a enviar, `fd`- El file descriptor sobre el que escribir.
Retorna: Nada.

- **ft_putendl_fd**: Envía una cadena al file descriptor dado, seguida de un salto de línea.
`void ft_putendl_fd(char *s, int fd);`
Parámetros: `s`- La cadena a enviar, `fd`- El file descriptor sobre el que escribir.
Retorna: Nada.

- **ft_putnbr_fd**: Envía un número entero al file descriptor especificado.
`void ft_putnbr_fd(int n, int fd);`
Parámetros: `n`- El número que se enviará, `fd`- El file descriptor sobre el que escribir.
Retorna: Nada.

Funciones de Listas Enlazadas

Para trabajar con listas enlazadas, es necesario definir la siguiente estructura de nodo en el archivo `libft.h`:

```
typedef struct s_list
{
    void *content;
    struct s_list *next;
} t_list;
```

Los miembros de la estructura `t_list` son:

- **content**: El contenido del nodo. El tipo `void *` permite guardar cualquier tipo de información.
- **next**: Un puntero al siguiente nodo de la lista o `NULL` si es el último nodo.

- **ft_lstnew** : Crea un nuevo nodo de lista.
`t_list *ft_lstnew(void *content);`
Parámetros: `content` - El contenido que se almacenará en el nuevo nodo.
Retorna: Un puntero al nuevo nodo creado.

- **ft_lstadd_front** : Añade un nodo al principio de la lista.
`void ft_lstadd_front(t_list **lst, t_list *new);`
Parámetros: `lst` - Dirección de un puntero al primer nodo de la lista, `new` - El nodo que se va a añadir al principio.
Retorna: Nada.

- **ft_lstsize** : Devuelve el número de nodos de la lista.
`int ft_lstsize(t_list *lst);`
Parámetros: `lst` - El primer nodo de la lista.
Retorna: La cantidad de nodos en la lista.

- **ft_lstlast** : Devuelve el último nodo de la lista.
`t_list *ft_lstlast(t_list *lst);`
Parámetros: `lst` - El primer nodo de la lista.
Retorna: El último nodo de la lista.

- **ft_lstadd_back** : Añade un nodo al final de la lista.
`void ft_lstadd_back(t_list **lst, t_list *new);`
Parámetros: `lst` - Dirección del primer nodo de la lista, `new` - El nodo que se añadirá al final.
Retorna: Nada.

- **ft_lstdelone** : Libera un nodo de la lista.
`void ft_lstdelone(t_list *lst, void (*del)(void *));`
Parámetros: `lst` - El nodo a liberar, `del` - Función que libera el contenido del nodo.
Retorna: Nada.
Ejemplo de funcion del, dentro del código se llamaría: `del(lst->content);`

```
void        del(void *lst)
{
    free(lst);
}
```

- **ft_lstclear** : Libera todos los nodos de la lista.
`void ft_lstclear(t_list **lst, void (*del)(void *));`
Parámetros: `lst` - Dirección de un puntero al primer nodo, `del` - Función que libera el contenido de cada nodo.
Retorna: Nada.

- **ft_lstiter** : Itera sobre cada nodo de la lista y aplica una función.
`void ft_lstiter(t_list *lst, void (*f)(void *));`
Parámetros: `lst` - El primer nodo de la lista, `f` - Función a aplicar sobre el contenido de cada nodo.
Retorna: Nada.
Ejemplo de funcion del, dentro del código se llamaría: `f(lst->content);`

```
void        f(void *content)
{
    content = "a";
}
```

- **ft_lstmap** : Crea una nueva lista aplicando una función a cada nodo de la lista original.
`t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));`
Parámetros: `lst` - El primer nodo de la lista, `f` - Función que se aplicará a cada nodo, `del` - Función que libera el contenido de un nodo.
Retorna: Una nueva lista con los resultados de aplicar `f` a cada nodo. Si falla la asignación de memoria, retorna `NULL`.