MEMSET

```
#include "stdio.h"

void    *ft_memset(void *b, int c, size_t len)
//DEVUELVE UN PUNTERO GENÉRICO VOID LLAMADO B

{

    size_t  i; // SIZE_T  Garantiza que el valor no sea negativo y se
ajuste al rango adecuado- 0 hasta 4,294,967,295 en sistemas de 32 bits
    unsigned char  *ptr; : se crea un puntero unsigned  char  *ptr para apuntar a la
ubicación de memoria a la que apunta el puntero void  *b.

// EL UNSIGNED CHAR es para asegurarse de que el valor de PTR se ajuste
al rango correcto de un byte (0 a 255) antes de asignarlo al bloque de
memoria.


     ptr = b; : esto se llama CASTEAR
// Convertimos el puntero genérico b a un puntero unsigned char
Dado que b es un puntero void, no se puede realizar operaciones de
indirección directamente sobre él(acceder al valor al que apunta el
puntero.), ya que el compilador no conoce el tamaño del tipo de datos
apuntado. Por lo tanto, se realiza un cast (conversión de tipos)
```

Por ejemplo, si tienes un puntero `int *p` que apunta a una variable `x`, puedes acceder al valor de `x` a través del puntero utilizando el operador de indirección `*`. Esto se hace de la siguiente manera: `*p` (donde `*p` representa el valor de `x`).

Sin embargo, en el código que proporcionaste, `b` es un puntero `void`, que es un tipo de puntero genérico que no tiene información sobre el tipo de datos al que apunta. Debido a esta falta de información de tipo, no se pueden realizar operaciones de indirección directamente sobre `b` para acceder o modificar los datos a los que apunta.

```
     i = 0;
     while (i < len)

    {
            ptr[i] = (unsigned char)c;
    // asignar el valor de c a la posición de memoria i dentro del
    bloque de memoria apuntado por ptr

    //El parámetro c se recibe como un entero (int), pero se realiza un
    casting explícito a (unsigned char) al asignar  ptr[i] = (unsigned char)c

    // EL UNSIGNED CHAR es para asegurarse de que el valor de c se
    ajuste al rango correcto de un byte (0 a 255) antes de asignarlo al
    bloque de memoria.
            i++;
        }     return (b);  }
```

:BZERO

```
void    ft_bzero(void *s, size_t n)
// La función ft_bzero no devuelve ningún valor, es de tipo void. Su propósito es establecer los primeros n bytes del bloque de memoria apuntado por s en cero (es decir, establecerlos en el valor binario de cero).

{
        unsigned char   *ptr;
        size_t i;

        ptr = (unsigned char *)s;
// se utiliza para asignar el valor del puntero s a la variable ptr, pero con un tipo de puntero diferente.
        i = 0;
        while (i < n)
        {
                i++;
                    *ptr = 0;
                    ptr++;
        }
}
```

:STRCHR

```
// EL INT C ES EL CARÁCTER QUE QUEREMOS BUSCAR, PASAMOS UN INT PARA QUE COGA EL MAYOR RANGO Y NO SE LIMITE AL CHAR EN RANGO. El tipo int se utiliza en lugar de char para permitir que se pase cualquier valor de caracter válido, incluyendo valores que están fuera del rango de char


char *ft_strchr(const char *s, int c)
{
    int i;

    i = 0;
    while (s[i] != '\0')
    {
        if (s[i] == (char)c)
//(CASTEO DE INT A CHAR) PARA BUSCAR EL CARÁCTER.
            return (&((char *)s)[i]);
//devuelve un puntero a la posición i en la cadena s mediante el uso de un casting a char *

        i++;
    }
    if ((char)c == '\0')
        return (&((char *)s)[i]);
//ESTO SOLO EN EL CASO DE QUE SE BUSCA EL CARÁCTER NULO, QUE SERÁ LA ÚLTIMA POSICIÓN \0

    return (0);
}
```

:MEMMOVE (sin superposición)

```
void    *ft_memmove(void *dst, const void *src, size_t len)
```

se indica que el puntero `src` puede apuntar a cualquier tipo de dato, ya que el tipo de dato no está especificado. Esto permite que la función `ft_memcpy` copie una secuencia de bytes desde la ubicación de memoria apuntada por `src` a la ubicación de memoria apuntada por `dst`, independientemente del tipo de dato.

Si se espera copiar solo caracteres, se puede cambiar `void` por `char`. Por ejemplo:

```
void *ft_memcpy(void *dst, const char *src, size_t n)
```

```
{
        size_t  i;

        i = 0;
        if (len)
        {
                if (dst < src)
                {
                        while (i < len)
                        {
                                ((unsigned char *)dst)[i] = ((unsigned char
*)src)[i];
                                i++;
                        }
                }
```

```
                else if (src < dst)
                {
                        while (len --)
```

La expresión `while (len--)` se utiliza para iterar desde `len` hasta 0, disminuyendo `len` en cada iteración

```
                                ((unsigned char *)dst)[len] = ((unsigned char
*)src)[len];
                }
        }
        return (dst);
}
```