3.5 Byte Manipulation Functions

There are two groups of functions that operate on multibyte fields, without interpreting the data, and without assuming that the data is a null-terminated C string. We need these types of functions when dealing with socket address structures because we need to manipulate fields such as IP addresses, which can contain bytes of 0, but are not C character strings. The functions beginning with `str` (for string), defined by including the `<string.h>` header, deal with null-terminated C character strings.

The first group of functions, whose names begin with `b` (for byte), are from 4.2BSD and are still provided by almost any system that supports the socket functions. The second group of functions, whose names begin with `mem` (for memory), are from the ANSI C standard and are provided with any system that supports an ANSI C library.

We first show the Berkeley-derived functions, although the only one we use in this text is `bzero`. (We use it because it has only two arguments and is easier to remember than the three-argument `memset` function, as explained on p. 8.) You may encounter the other two functions, `bcopy` and `bcmp`, in existing applications.

```
#include <strings.h>

void bzero(void *dest, size_t nbytes);

void bcopy(const void *src, void *dest, size_t nbytes);

int bcmp(const void *ptr1, const void *ptr2, size_t nbytes);
```

Returns: 0 if equal, nonzero if unequal

> This is our first encounter with the ANSI C `const` qualifier. In the three uses here, it indicates that what is pointed to by the pointer with this qualification, *src, ptr1*, and *ptr2*, is not modified by the function. Worded another way, the memory pointed to by the `const` pointer is read but not modified by the function.

`bzero` sets the specified number of bytes to 0 in the destination. We often use this function to initialize a socket address structure to 0. `bcopy` moves the specified number of bytes from the source to the destination. `bcmp` compares two arbitrary byte strings. The return value is zero if the two byte strings are identical; otherwise, it is nonzero.

The following functions are the ANSI C functions:

```
#include <string.h>

void *memset(void *dest, int c, size_t len);

void *memcpy(void *dest, const void *src, size_t nbytes);

int memcmp(const void *ptr1, const void *ptr2, size_t nbytes);
```

Returns: 0 if equal, <0 or >0 if unequal (see text)

`memset` sets the specified number of bytes to the value *c* in the destination. `memcpy` is similar to `bcopy`, but the order of the two pointer arguments is swapped. `bcopy` correctly handles overlapping fields, while the behavior of `memcpy` is undefined if the source and destination overlap. The ANSI C `memmove` function must be used when the fields overlap.

> One way to remember the order of the two pointers for `memcpy` is to remember that they are written in the same left-to-right order as an assignment statement in C:
>
> *dest* = *src*;
>
> One way to remember the order of the final two arguments to `memset` is to realize that all of the ANSI C `memXXX` functions require a length argument, and it is always the final argument.

`memcmp` compares two arbitrary byte strings and returns 0 if they are identical. If not identical, the return value is either greater than 0 or less than 0, depending on whether the first unequal byte pointed to by *ptr1* is greater than or less than the corresponding byte pointed to by *ptr2*. The comparison is done assuming the two unequal bytes are `unsigned chars`.