Section 13.5 gives examples of how strcpy and strncpy are typically used. Although neither function is completely safe, strncpy at least provides a way to limit the number of characters it will copy.

Concatenation Functions

```
char *strcat(char * restrict s1,
             const char * restrict s2);
char *strncat(char * restrict s1,
              const char * restrict s2, size_t n);
```

strcat strcat appends its second argument to the end of the first argument. Both arguments must be null-terminated strings; strcat puts a null character at the end of the concatenated string. Consider the following example:

```
char str[7] = "tea";

strcat(str, "bag");    /* adds b, a, g, \0 to end of str */
```

The letter b overwrites the null character after the a in "tea", so that str now contains the string "teabag". strcat returns its first argument (a pointer).

strncat strncat is the same as strcat, except that its third argument limits the number of characters it will copy:

```
char str[7] = "tea";

strncat(str, "bag", 2);    /* adds b, a, \0 to str    */
strncat(str, "bag", 3);    /* adds b, a, g, \0 to str */
strncat(str, "bag", 4);    /* adds b, a, g, \0 to str */
```

As these examples show, strncat always leaves the resulting string properly null-terminated.

In Section 13.5, we saw that a call of strncat often has the following appearance:

```
strncat(str1, str2, sizeof(str1) - strlen(str1) - 1);
```

The third argument calculates the amount of space remaining in str1 (given by the expression sizeof(str1) - strlen(str1)) and then subtracts 1 to ensure that there will be room for the null character.

Comparison Functions

```
int memcmp(const void *s1, const void *s2, size_t n);
int strcmp(const char *s1, const char *s2);
int strcoll(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2,
            size_t n);
size_t strxfrm(char * restrict s1,
               const char * restrict s2, size_t n);
```