

## Output Functions

```
int sprintf(char * restrict s,
            const char * restrict format, ...);
int snprintf(char * restrict s, size_t n,
            const char * restrict format, ...);
```

*Note:* In this and subsequent chapters, the prototype for a function that is new in C99 will be in italics. Also, the name of the function will be italicized when it appears in the left margin.

**sprintf** The `sprintf` function is similar to `printf` and `fprintf`, except that it writes output into a character array (pointed to by its first argument) instead of a stream. `sprintf`'s second argument is a format string identical to that used by `printf` and `fprintf`. For example, the call

```
sprintf(date, "%d/%d/%d", 9, 20, 2010);
```

will write "9/20/2010" into `date`. When it's finished writing into a string, `sprintf` adds a null character and returns the number of characters stored (not counting the null character). If an encoding error occurs (a wide character could not be translated into a valid multibyte character), `sprintf` returns a negative value.

`sprintf` has a variety of uses. For example, we might occasionally want to format data for output without actually writing it. We can use `sprintf` to do the formatting, then save the result in a string until it's time to produce output. `sprintf` is also convenient for converting numbers to character form.

**snprintf** The `snprintf` function is the same as `sprintf`, except for the additional parameter `n`. No more than `n - 1` characters will be written to the string, not counting the terminating null character, which is always written unless `n` is zero. (Equivalently, we could say that `snprintf` writes at most `n` characters to the string, the last of which is a null character.) For example, the call

```
snprintf(name, 13, "%s, %s", "Einstein", "Albert");
```

will write "Einstein, Al" into `name`.

`snprintf` returns the number of characters that would have been written (not including the null character) had there been no length restriction. If an encoding error occurs, `snprintf` returns a negative number. To see if `snprintf` had room to write all the requested characters, we can test whether its return value was nonnegative and less than `n`.

## Input Functions

```
int sscanf(const char * restrict s,
          const char * restrict format, ...);
```