assumes that the arguments are of type `int`. Functions such as `printf` and `scanf` rely on the format string, which describes the number of additional arguments and the type of each.

Another problem has to do with passing `NULL` as an argument. `NULL` is usually defined to represent `0`. When `0` is passed to a function with a variable argument list, the compiler assumes that it represents an integer—there's no way it can tell that we want it to represent the null pointer. The solution is to add a cast, writing `(void *) NULL` or `(void *) 0` instead of `NULL`. (See the Q&A section at the end of Chapter 17 for more discussion of this point.)

The v...printf Functions

```
int vfprintf(FILE * restrict stream,
             const char * restrict format,
             va_list arg);                    from <stdio.h>
int vprintf(const char * restrict format,
            va_list arg);                     from <stdio.h>
int vsnprintf(char * restrict s, size_t n,
              const char * restrict format,
              va_list arg);                   from <stdio.h>
int vsprintf(char * restrict s,
             const char * restrict format,
             va_list arg);                    from <stdio.h>
```

vfprintf
vprintf
vsprintf

C99

The `vfprintf`, `vprintf`, and `vsprintf` functions (the "v...`printf` functions") belong to `<stdio.h>`. We're discussing them in this section because they're invariably used in conjunction with the macros in `<stdarg.h>`. C99 adds the `vsnprintf` function.

The v...`printf` functions are closely related to `fprintf`, `printf`, and `sprintf`. Unlike these functions, however, the v...`printf` functions have a fixed number of arguments. Each function's last argument is a `va_list` value, which implies that it will be called by a function with a variable argument list. In practice, the v...`printf` functions are used primarily for writing "wrapper" functions that accept a variable number of arguments, which are then passed to a v...`printf` function.

As an example, let's say that we're working on a program that needs to display error messages from time to time. We'd like each message to begin with a prefix of the form

`** Error` *n*`:`

where *n* is `1` for the first error message and increases by one for each subsequent error. To make it easier to produce error messages, we'll write a function named `errorf` that's similar to `printf`, but adds `** Error` *n*`:` to the beginning of