The ...printf Functions

```
int fprintf(FILE * restrict stream,
            const char * restrict format, ...);
int printf(const char * restrict format, ...);
```

fprintf
printf

The `fprintf` and `printf` functions write a variable number of data items to an output stream, using a format string to control the appearance of the output.

ellipsis ➤26.1

The prototypes for both functions end with the `...` symbol (an *ellipsis*), which indicates a variable number of additional arguments. Both functions return the number of characters written: a negative return value indicates that an error occurred.

The only difference between `printf` and `fprintf` is that `printf` always writes to `stdout` (the standard output stream), whereas `fprintf` writes to the stream indicated by its first argument:

```
printf("Total: %d\n", total);          /* writes to stdout */
fprintf(fp, "Total: %d\n", total);     /* writes to fp */
```

A call of `printf` is equivalent to a call of `fprintf` with `stdout` as the first argument.

Don't think of `fprintf` as merely a function that writes data to disk files, though. Like many functions in `<stdio.h>`, `fprintf` works fine with any output stream. In fact, one of the most common uses of `fprintf`—writing error messages to `stderr`, the standard error stream—has nothing to do with disk files. Here's what such a call might look like:

```
fprintf(stderr, "Error: data file can't be opened.\n");
```

Writing the message to `stderr` guarantees that it will appear on the screen even if the user redirects `stdout`.

v...printf functions ➤26.1

There are two other functions in `<stdio.h>` that can write formatted output to a stream. These functions, named `vfprintf` and `vprintf`, are fairly obscure. Both rely on the `va_list` type, which is declared in `<stdarg.h>`, so they're discussed along with that header.

...printf Conversion Specifications

Both `printf` and `fprintf` require a format string containing ordinary characters and/or conversion specifications. Ordinary characters are printed as is; conversion specifications describe how the remaining arguments are to be converted to character form for display. Section 3.1 described conversion specifications briefly. and we added more details in later chapters. We'll now review what we know about conversion specifications and fill in the remaining gaps.

A `...printf` conversion specification consists of the `%` character. followed by as many as five distinct items: