fputs

`fputs` is a more general version of `puts`. Its second argument indicates the stream to which the output should be written:

```
fputs("Hi, there!", fp);   /* writes to fp */
```

Unlike `puts`, the `fputs` function doesn't write a new-line character unless one is present in the string.

Both functions return `EOF` if a write error occurs; otherwise, they return a nonnegative number.

Input Functions

```
char *fgets(char * restrict s, int n,
            FILE * restrict stream);
char *gets(char *s);
```

gets

The `gets` function, which we first encountered in Section 13.3, reads a line of input from `stdin`:

```
gets(str);   /* reads a line from stdin */
```

`gets` reads characters one by one, storing them in the array pointed to by `str`, until it reads a new-line character (which it discards).

fgets

`fgets` is a more general version of `gets` that can read from any stream. `fgets` is also safer than `gets`, since it limits the number of characters that it will store. Here's how we might use `fgets`, assuming that `str` is the name of a character array:

```
fgets(str, sizeof(str), fp);   /* reads a line from fp */
```

This call will cause `fgets` to read characters until it reaches the first new-line character or `sizeof(str) - 1` characters have been read, whichever happens first. If it reads the new-line character, `fgets` stores it along with the other characters. (Thus, `gets` *never* stores the new-line character, but `fgets` *sometimes* does.)

Both `gets` and `fgets` return a null pointer if a read error occurs or they reach the end of the input stream before storing any characters. (As usual, we can call `feof` or `ferror` to determine which situation occurred.) Otherwise, both return their first argument, which points to the array in which the input was stored. As you'd expect, both functions store a null character at the end of the string.

Now that you know about `fgets`, I'd suggest using it instead of `gets` in most situations. With `gets`, there's always the possibility of stepping outside the bounds of the receiving array, so it's safe to use only when the string being read is *guaranteed* to fit into the array. When there's no guarantee (and there usually isn't), it's much safer to use `fgets`. Note that `fgets` will read from the standard input stream if passed `stdin` as its third argument:

```
fgets(str, sizeof(str), stdin);
```