All six functions treated this string as a valid number, although the integer functions stopped at the decimal point. The `strtol` and `strtoul` functions were able to indicate that they didn't completely consume the string.

If `foo` is the command-line argument, the output will be

```
Function   Return Value
--------   ------------
atof       0
atoi       0
atol       0

Function   Return Value   Valid?   String Consumed?
--------   ------------   ------   ----------------
strtod     0                Yes            No
strtol     0                Yes            No
strtoul    0                Yes            No
```

All the functions looked at the letter `f` and immediately returned zero. The `str...` functions didn't change `errno`, but we can tell that something went wrong from the fact that the functions didn't consume the string.

## Pseudo-Random Sequence Generation Functions

```
int rand(void);
void srand(unsigned int seed);
```

The `rand` and `srand` functions support the generation of pseudo-random numbers. These functions are useful in simulation programs and game-playing programs (to simulate a dice roll or the deal in a card game, for example).

rand

Each time it's called, `rand` returns a number between 0 and `RAND_MAX` (a macro defined in `<stdlib.h>`). The numbers returned by `rand` aren't actually random; they're generated from a "seed" value. To the casual observer, however, `rand` appears to produce an unrelated sequence of numbers.

srand

Calling `srand` supplies the seed value for `rand`. If `rand` is called prior to `srand`, the seed value is assumed to be 1. Each seed value determines a particular sequence of pseudo-random numbers; `srand` allows us to select which sequence we want.

A program that always uses the same seed value will always get the same sequence of numbers from `rand`. This property can sometimes be useful: the program behaves the same way each time it's run, making testing easier. However, we usually want `rand` to produce a *different* sequence each time the program is run. (A poker-playing program that always deals the same cards isn't likely to be popular.) The easiest way to "randomize" the seed values is to call the `time` function, which returns a number that encodes the current date and time. Passing `time`'s return value to `srand` makes the behavior of `rand` vary from one run to the next. See the `guess.c` and `guess2.c` programs (Section 10.2) for examples of this technique.

`time` function ➤26.3