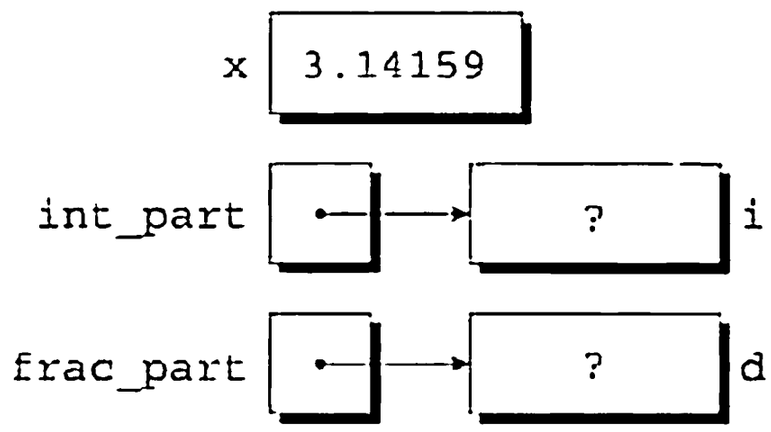
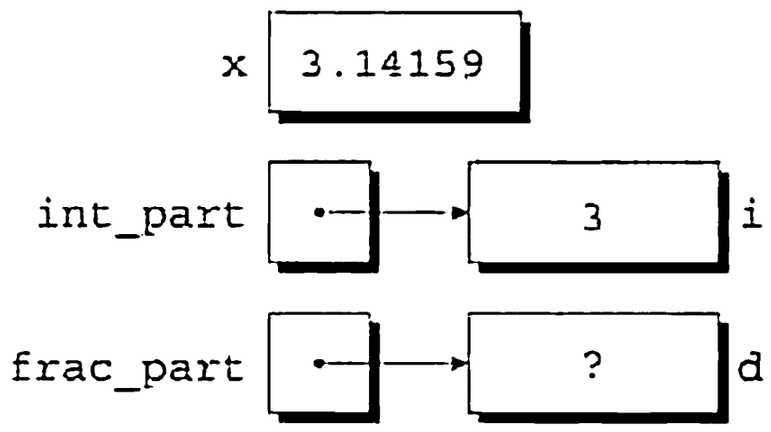
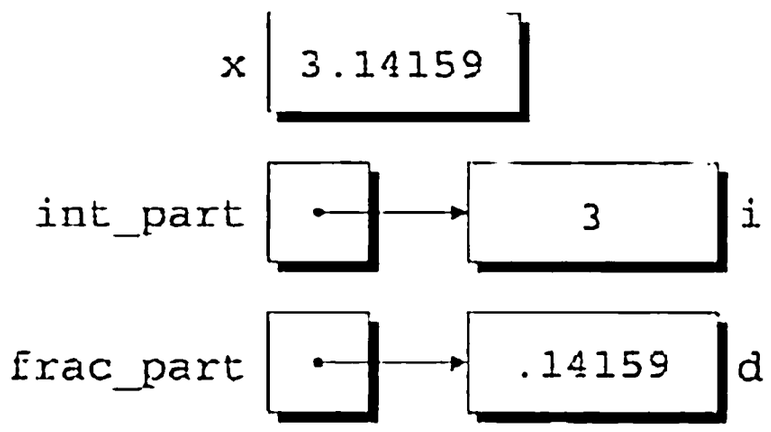


The first assignment in the body of `decompose` converts the value of `x` to type `long` and stores it in the object pointed to by `int_part`. Since `int_part` points to `i`, the assignment puts the value 3 in `i`:



The second assignment fetches the value that `int_part` points to (the value of `i`), which is 3. This value is converted to type `double` and subtracted from `x`, giving .14159, which is then stored in the object that `frac_part` points to:



When `decompose` returns, `i` and `d` will have the values 3 and .14159, just as we originally wanted.

Using pointers as arguments to functions is actually nothing new; we've been doing it in calls of `scanf` since Chapter 2. Consider the following example:

```
int i;
...
scanf("%d", &i);
```

We must put the `&` operator in front of `i` so that `scanf` is given a *pointer* to `i`; that pointer tells `scanf` where to put the value that it reads. Without the `&`, `scanf` would be supplied with the *value* of `i`.

Although `scanf`'s arguments must be pointers, it's not always true that every argument needs the `&` operator. In the following example, `scanf` is passed a pointer variable: