You use the right-left rule, sort of like peeling an onion: you start with the name, and read to the right until you can't, then you move left until you can't, and then move right again. Nothing like a perverse example to demonstrate the point:

```
const int *(*f[5])(int *, char []);
```

Using the right-left rule, you get:

- locate `f`, then move right, so `f` is an array of 5...
- moving left, `f` is an array of 5 pointers...
- moving right, `f` is an array of 5 pointers to a function...
- continue to move right, `f` is an array of 5 pointers to a function with two arguments (we can skip ahead and read the function prototype later)...
- moving left, `f` is an array of 5 pointers to function with two arguments that returns a pointer to...
- moving left, `f` is an array of 5 pointers to function with two arguments that returns a pointer to `int`...
- moving left for the last time, `f` is an array of 5 pointers to function with two arguments that returns a pointer to `const int`.

You can of course also use the right-left rule to write declarations. In the example, the type qualifier `const` is also used. There are two type qualifiers: `const` (object is read only) or `volatile` (object may change in unexpected ways).

`volatile` is for decorating an object that may be changed by an asynchronous process -- e.g., a global variable that is updated by an interrupt handler. Marking such variables as `volatile` tells the compilers not to cache the accessed values.