volatile type qualifier ➤20.3

If that's not enough to convince you, consider this: If a `const` object is also declared to be `volatile`, its value may change at any time during execution. Here's an example from the C standard:

```
extern const volatile int real_time_clock;
```

The `real_time_clock` variable may not be changed by the program (because it's declared `const`), yet its value may change via some other mechanism (because it's declared `volatile`).

Q: Why is the syntax of declarators so odd?

A: Declarations are intended to mimic use. A pointer declarator has the form `*p`, which matches the way the indirection operator will later be applied to `p`. An array declarator has the form `a[...]`, which matches the way the array will later be subscripted. A function declarator has the form `f(...)`, which matches the syntax of a function call. This reasoning extends to even the most complicated declarators. Consider the `file_cmd` array of Section 17.7, whose elements are pointers to functions. The declarator for `file_cmd` has the form

```
(*file_cmd[])(void)
```

and a call of one of the functions has the form

```
(*file_cmd[n])();
```

The parentheses, brackets, and * are in identical positions.

Exercises

Section 18.1

1. For each of the following declarations, identify the storage class, type qualifiers, type specifiers, declarators, and initializers.

   (a) `static char **lookup(int level);`
   (b) `volatile unsigned long io_flags;`
   (c) `extern char *file_name[MAX_FILES], path[];`
   (d) `static const char token_buf[] = "";`

Section 18.2

W 2. Answer each of the following questions with `auto`, `extern`, `register`, and/or `static`.

   (a) Which storage class is used primarily to indicate that a variable or function can be shared by several files?
   (b) Suppose that a variable `x` is to be shared by several functions in one file but hidden from functions in other files. Which storage class should `x` be declared to have?
   (c) Which storage classes can affect the storage duration of a variable?

3. List the storage duration (static or automatic), scope (block or file), and linkage (internal, external, or none) of each variable and parameter in the following file:

```
extern float a;

void f(register double b)
{
  static int c;
  auto char d;
}
```

W 4. Let `f` be the following function. What will be the value of `f(10)` if `f` has never been called before? What will be the value of `f(10)` if `f` has been called five times previously?

```
int f(int i)
{
  static int j = 0;
  return i * j++;
}
```

5. State whether each of the following statements is true or false. Justify each answer.
   (a) Every variable with static storage duration has file scope.
   (b) Every variable declared inside a function has no linkage.
   (c) Every variable with internal linkage has static storage duration.
   (d) Every parameter has block scope.

6. The following function is supposed to print an error message. Each message is preceded by an integer, indicating the number of times the function has been called. Unfortunately, the function always displays 1 as the number of the error message. Locate the error and show how to fix it without making any changes outside the function.

```
void print_error(const char *message)
{
  int n = 1;
  printf("Error %d: %s\n", n++, message);
}
```

Section 18.3

7. Suppose that we declare `x` to be a `const` object. Which one of the following statements about `x` is *false*?
   (a) If `x` is of type `int`, it can be used as the value of a case label in a `switch` statement.
   (b) The compiler will check that no assignment is made to `x`.
   (c) `x` is subject to the same scope rules as variables.
   (d) `x` can be of any type.

Section 18.4

W 8. Write a complete description of the type of `x` as specified by each of the following declarations.
   (a) `char (*x[10])(int);`
   (b) `int (*x(int))[5];`
   (c) `float *(*x(void))(int);`
   (d) `void (*x(int, void (*y)(int)))(int);`

9. Use a series of type definitions to simplify each of the declarations in Exercise 8.

W 10. Write declarations for the following variables and functions:
   (a) `p` is a pointer to a function with a character pointer argument that returns a character pointer.