can simply move the declaration of i outside f so that i has file scope.) What's confusing about this entire business is that each declaration or definition of i establishes a different scope; sometimes it's file scope, and sometimes it's block scope.

*Q: **Why can't const objects be used in constant expressions? const means "constant," right? [p. 466]**

A: In C, const means "read-only," not "constant." Let's look at a few examples that illustrate why const objects can't be used in constant expressions.

To start with, a const object might only be constant during its *lifetime*, not throughout the execution of the program. Suppose that a const object is declared inside a function:

```
void f(int n)
{
  const int m = n / 2;
  ...
}
```

When f is called, m will be initialized to the value of n / 2. The value of m will then remain constant until f returns. When f is called the next time, m will likely be given a different value. That's where the problem arises. Suppose that m appears in a switch statement:

```
void f(int n)
{
  const int m = n / 2;
  ...
  switch (...) {
    ...
    case m: ...    /*** WRONG ***/
    ...
  }
  ...
}
```

The value of m won't be known until f is called, which violates C's rule that the values of case labels must be constant expressions.

Next, let's look at const objects declared outside blocks. These objects have external linkage and can be shared among files. If C allowed the use of const objects in constant expressions, we could easily find ourselves in the following situation:

```
extern const int n;
int a[n];    /*** WRONG ***/
```

n is probably defined in another file, making it impossible for the compiler to determine a's length. (I'm assuming that a is an external variable, so it can't be a variable-length array.)