The creators of C99 were unhappy with this state of affairs, because programmers were unlikely to expect that simply adding braces within an if statement would cause undefined behavior. To avoid the problem, they decided that the inner statements would always be considered blocks. As a result, Example 1 and Example 2 are equivalent, with both exhibiting undefined behavior.

A similar problem can arise when a compound literal is part of the controlling expression of a selection statement or iteration statement. For this reason, each entire selection statement and iteration statement is considered to be a block as well (as though an invisible set of braces surrounds the entire statement). So, for example, an if statement with an else clause consists of three blocks: each of the two inner statements is a block, as is the entire if statement.

Q: **You said that storage for a variable with automatic storage duration is allocated when the surrounding block is executed. Is this true for C99's variable-length arrays? [p. 460]**

A: No. Storage for a variable-length array isn't allocated at the beginning of the surrounding block, because the length of the array isn't yet known. Instead, it's allocated when the declaration of the array is reached during the execution of the block. In this respect, variable-length arrays are different from all other automatic variables.

Q: **What exactly is the difference between "scope" and "linkage"? [p. 460]**

A: Scope is for the benefit of the compiler, while linkage is for the benefit of the linker. The compiler uses the scope of an identifier to determine whether or not it's legal to refer to the identifier at a given point in a file. When the compiler translates a source file into object code, it notes which names have external linkage, eventually storing these names in a table inside the object file. Thus, the linker has access to names with external linkage; names with internal linkage or no linkage are invisible to the linker.

Q: **I don't understand how a name could have block scope but external linkage. Could you elaborate? [p. 463]**

A: Certainly. Suppose that one source file defines a variable i:

```
int i;
```

Let's assume that the definition of i lies outside any function, so i has external linkage by default. In another file. there's a function f that needs to access i, so the body of f declares i as extern:

```
void f(void)
{
  extern int i;
  ...
}
```

In the first file, i has file scope. Within f, however, i has block scope. If other functions besides f need access to i, they'll need to declare it separately. (Or we