program uses only the particular variable in scope at the moment, so assigning the value 113 to the texas in the inner block in oil() has no effect on the other variables of the same name.

Let's summarize the sequence of events. When main() starts, the program allocates space for texas and year, and these variables come into scope. When the program calls oil(), these variables remain in memory but pass out of scope. Two new variables, x and texas, are allocated and come into scope. When program execution reaches the inner block in oil(), the new texas passes out of scope as it is superseded by an even newer definition. The variable x, however, stays in scope because the block doesn't define a new x. When execution exits the block, the memory for the newest texas is freed, and texas number 2 comes back into scope. When the oil() function terminates, that texas and x expire, and the original texas and year come back into scope.

Incidentally, you can use the C++ (and C) keyword auto to indicate the storage class explicitly:

```cpp
int froob(int n)
{
    auto float ford;
    ...
}
```

Because you can use the auto keyword only with variables that already are automatic by default, programmers rarely bother using it. Occasionally, it's used to clarify code to the reader. For example, you can use it to indicate that you purposely are creating an automatic variable that overrides a global definition, such as those we discuss shortly.

**Initialization of Automatic Variables**

You can initialize an automatic variable with any expression whose value will be known when the declaration is reached:

```cpp
int w;       // value of w is indeterminate
int x = 5;   // initialized with a constant expression
int y = 2 * x; // use previously determined value of x
cin >> w;
```