linkage, internal linkage, and no linkage. All three last for the duration of the program; they are less ephemeral than automatic variables. Because the number of static variables doesn't change as the program runs, the program doesn't need a special device like a stack to manage them. Instead, the compiler allocates a fixed block of memory to hold all the static variables, and those variables stay present as long as the program executes. Also, if you don't explicitly initialize a static variable, the compiler sets it to zero. Static arrays and structures have all the bits of each element or member set to zero by default.

**Compatibility Note**

Classic K&R C did not allow you to initialize automatic arrays and structures, but it did allow you to initialize static arrays and structures. ANSI C and C++ allow you to initialize both kinds. But some older C++ translators use C compilers that are not fully ANSI C-compliant. If you are using such an implementation, you might need to use one of the three varieties of static storage classes for initializing arrays and structures.

Let's see how to create the three different kinds of static duration variables; then we can go on to examine their properties. To create a static duration variable with external linkage, declare it outside of any block. To create a static duration variable with internal linkage, declare it outside of any block and use the `static` storage class modifier. To create a static duration variable with no linkage, declare it inside a block using the `static` modifier. The following code fragment shows these three variations:

```c
...
int global = 1000;       // static duration, external linkage
static int one_file = 50;  // static duration, internal linkage
int main()
{
    ...
}
void funct1(int n)
{
    static int count = 0;  // static duration, no linkage
    int llama = 0;
```