bly wouldn't deliberately reuse a function name for some other purpose, it can be hard to avoid in large programs. An excessive number of names with external linkage can result in what C programmers call "name space pollution": names in different files accidentally conflicting with each other. Using `static` helps prevent this problem.

Function parameters have the same properties as `auto` variables: automatic storage duration, block scope, and no linkage. The only storage class that can be specified for parameters is `register`.

Summary

Now that we've covered the various storage classes, let's summarize what we know. The following program fragment shows all possible ways to include—or omit—storage classes in declarations of variables and parameters.

```
int a;
extern int b;
static int c;

void f(int d, register int e)
{
  auto int g;
  int h;
  static int i;
  extern int j;
  register int k;
}
```

Table 18.1 shows the properties of each variable and parameter in this example.

Table 18.1
Properties of Variables and Parameters

Name	*Storage Duration*	*Scope*	*Linkage*
`a`	static	file	external
`b`	static	file	†
`c`	static	file	internal
`d`	automatic	block	none
`e`	automatic	block	none
`g`	automatic	block	none
`h`	automatic	block	none
`i`	static	block	none
`j`	static	block	†
`k`	automatic	block	none

†The definitions of `b` and `j` aren't shown, so it's not possible to determine the linkage of these variables. In most cases, the variables will be defined in another file and will have external linkage.

Of the four storage classes, the most important are `static` and `extern`. `auto` has no effect, and modern compilers have made `register` less important.