sends you an error message, for you can have only one
definition for an external function. If it fails to find any
definition in your files, the function then searches the
libraries. This implies that if you define a function having
the same name as a library function, the compiler uses
your version rather than the library version. (However, C++
reserves the names of the standard library functions,
meaning you shouldn't reuse them.) Some compiler-linkers
need explicit instructions to identify which libraries to
search.

**Language Linking**

There is another form of linking, called *language linking*, that affects functions. First, a little
background. A linker needs a different symbolic name for each distinct function. In C, this
was simple to implement because there could only be one C function by a given name. So,
for internal purposes, a C compiler might translate a C function name such as `spiff` to
`_spiff`.

The C approach is termed *C language linkage*. C++, however, can have several functions
with the same C++ name that have to be translated to separate symbolic names. Thus, the
C++ compiler indulged in the process of name mangling or name decoration (as discussed in
Chapter 8) to generate different symbolic names for overloaded functions. For example,
it could convert `spiff(int)` to, say, `_spiff_i`, and `spiff(double, double)` to `_spiff_d_d`. The
C++ approach is *C++ language linkage*.

When the linker looks for a function to match a C++ function call, it uses a different look-up
method than it does to match a C function call. But suppose you want to use a precompiled
function from a C library in a C++ program? For example, suppose you have this code:

```c
spiff(22); // want spiff(int) from a C library
```

Its symbolic name in the C library file is `_spiff`, but, for our hypothetical linker, the C++
look-up convention is to look for the symbolic name `_spiff_i`. To get around this problem,
you can use the function prototype to indicate which protocol to use:

```c
extern "C" void spiff(int); // use C protocol for name look-up
```