tion access an external variable in another file? How can two files share the same macro definition or type definition? The answer lies with the `#include` directive, which makes it possible to share information—function prototypes, macro definitions, type definitions, and more—among any number of source files.

The `#include` directive tells the preprocessor to open a specified file and insert its contents into the current file. Thus, if we want several source files to have access to the same information, we'll put that information in a file and then use `#include` to bring the file's contents into each of the source files. Files that are included in this fashion are called ***header files*** (or sometimes ***include files***); I'll discuss them in more detail later in this section. By convention, header files have the extension `.h`.

Note: The C standard uses the term "source file" to refer to all files written by the programmer, including both `.c` and `.h` files. I'll use "source file" to refer to `.c` files only.

The `#include` Directive

The `#include` directive has two primary forms. The first form is used for header files that belong to C's own library:

`#include` directive (form 1)

```
#include <filename>
```

The second form is used for all other header files, including any that we write:

`#include` directive (form 2)

```
#include "filename"
```

Q&A

The difference between the two is a subtle one having to do with how the compiler locates the header file. Here are the rules that most compilers follow:

- `#include <`*filename*`>`: Search the directory (or directories) in which system header files reside. (On UNIX systems, for example, system header files are usually kept in the directory `/usr/include`.)
- `#include "`*filename*`"`: Search the current directory, then search the directory (or directories) in which system header files reside.

The places to be searched for header files can usually be altered, often by a command-line option such as `-I`*path*.

Don't use brackets when including header files that you have written:

```
#include <myheader.h>    /*** WRONG ***/
```

The preprocessor will probably look for `myheader.h` where the system header files are kept (and, of course, won't find it).
