The file name in an `#include` directive may include information that helps locate the file, such as a directory path or drive specifier:

```
#include "c:\cprogs\utils.h"   /* Windows path */

#include "/cprogs/utils.h"     /* UNIX path */
```

Although the quotation marks in the `#include` directive make file names look like string literals, the preprocessor doesn't treat them that way. (That's fortunate, since `\c` and `\u`—which appear in the Windows example—would be treated as escape sequences in a string literal.)

portability tip

It's usually best not to include path or drive information in `#include` *directives. Such information makes it difficult to compile a program when it's transported to another machine or, worse, another operating system.*

For example, the following Windows `#include` directives specify drive and/or path information that may not always be valid:

```
#include "d:utils.h"
#include "\cprogs\include\utils.h"
#include "d:\cprogs\include\utils.h"
```

The following directives are better: they don't mention specific drives, and paths are relative rather than absolute:

```
#include "utils.h"
#include "..\include\utils.h"
```

The `#include` directive has a third form that's used less often than the other two:

`#include` directive (form 3)

`#include` *tokens*

preprocessing tokens ➤ *14.3*

where *tokens* is any sequence of preprocessing tokens. The preprocessor will scan the tokens and replace any macros that it finds. After macro replacement, the resulting directive must match one of the other forms of `#include`. The advantage of the third kind of `#include` is that the file name can be defined by a macro rather than being "hard-coded" into the directive itself, as the following example shows:

```
#if defined(IA32)
  #define CPU_FILE "ia32.h"
#elif defined(IA64)
  #define CPU_FILE "ia64.h"
#elif defined(AMD64)
  #define CPU_FILE "amd64.h"
#endif

#include CPU_FILE
```