```
#define STR_LEN 80
#define TRUE    1
#define FALSE   0
#define PI      3.14159
#define CR      '\r'
#define EOS     '\0'
#define MEM_ERR "Error: not enough memory"
```

Using `#define` to create names for constants has several significant advantages:

- *It makes programs easier to read.* The name of the macro—if well-chosen—helps the reader understand the meaning of the constant. The alternative is a program full of "magic numbers" that can easily mystify the reader.
- *It makes programs easier to modify.* We can change the value of a constant throughout a program by modifying a single macro definition. "Hard-coded" constants are more difficult to change, especially since they sometimes appear in a slightly altered form. (For example, a program with an array of length 100 may have a loop that goes from 0 to 99. If we merely try to locate occurrences of `100` in the program, we'll miss the `99`.)
- *It helps avoid inconsistencies and typographical errors.* If a numerical constant like `3.14159` appears many times in a program, chances are it will occasionally be written `3.1416` or `3.14195` by accident.

Although simple macros are most often used to define names for constants, they do have other applications:

- *Making minor changes to the syntax of C.* We can—in effect—alter the syntax of C by defining macros that serve as alternate names for C symbols. For example, programmers who prefer Pascal's `begin` and `end` to C's { and } can define the following macros:

  ```
  #define BEGIN {
  #define END   }
  ```

  We could go so far as to invent our own language. For example, we might create a `LOOP` "statement" that establishes an infinite loop:

  ```
  #define LOOP for (;;)
  ```

  Changing the syntax of C usually isn't a good idea, though, since it can make programs harder for others to understand.
- *Renaming types.* In Section 5.2, we created a Boolean type by renaming `int`:

  ```
  #define BOOL int
  ```

  type definitions ➤7.5

  Although some programmers use macros for this purpose, type definitions are a superior way to define type names.
- *Controlling conditional compilation.* Macros play an important role in controlling conditional compilation, as we'll see in Section 14.4. For example, the presence of the following line in a program might indicate that it's to be com-