There's no longer a need to declare the variable *ch,* so that line is gone, as are the curly braces belonging to `while`.

Save the changes to disk as TYPER3.C. Compile and run.

The output behaves the same — no surprises there. But, the code is much tighter (albeit a little less readable).

TIP

- Despite this trick, it's often better to write your code in several steps rather than combine things on one line, "inside out." By illustrating the several steps, you make your code more readable.
- Write things out long-ways first. Then, after you're certain that the code works, think about recoding from the inside out.
- If you do use this trick at any length, be sure to make use of comments to describe what your thinking is. That helps later, in case you ever need to debug your code.

Not to Beat a Dead Horse or Anything. . . .

It has come to this — your first inane programmer joke in C:

```
while(dead_horse)
      beat();
```

Drawing on your vast knowledge of C, you can now appreciate what humor there is in the "no use beating a dead horse" cliché translated into the C programming language. Here are the specifics — if you can hold your sides in long enough:

- `dead_horse` is the name of a variable whose value is either TRUE or FALSE. While `dead_horse` is TRUE, the loop repeats.
- I have also seen the first line written this way:

  ```
  while(horse==dead)
  and also
  while(!alive)
  ```

  In other words, "While it's a dead horse. . . ."
- The `beat()` function is repeated over and over as long as the value of `dead_horse` is TRUE (or as long as `horse==dead`, in the alternative form).
- You don't have to enclose `beat()` in curly braces because it's the only statement belonging to the loop.
- Yuck. Yuck. Yuck.