

- If an external variable is assigned an incorrect value, it may be difficult to identify the guilty function. It's like trying to solve a murder committed at a crowded party—there's no easy way to narrow the list of suspects.
- Functions that rely on external variables are hard to reuse in other programs. A function that depends on external variables isn't self-contained; to reuse the function, we'll have to drag along any external variables that it needs.

Many C programmers rely far too much on external variables. One common abuse: using the same external variable for different purposes in different functions. Suppose that several functions need a variable named `i` to control a `for` statement. Instead of declaring `i` in each function that uses it, some programmers declare it at the top of the program, thereby making the variable visible to all functions. This practice is poor not only for the reasons listed earlier, but also because it's misleading; someone reading the program later may think that the uses of the variable are related, when in fact they're not.

When you use external variables, make sure they have meaningful names. (Local variables don't always need meaningful names: it's often hard to think of a better name than `i` for the control variable in a `for` loop.) If you find yourself using names like `i` and `temp` for external variables, that's a clue that perhaps they should really be local variables.



Making variables external when they should be local can lead to some rather frustrating bugs. Consider the following example, which is supposed to display a 10 × 10 arrangement of asterisks:

```
int i;

void print_one_row(void)
{
    for (i = 1; i <= 10; i++)
        printf("*");
}

void print_all_rows(void)
{
    for (i = 1; i <= 10; i++) {
        print_one_row();
        printf("\n");
    }
}
```

Instead of printing 10 rows, `print_all_rows` prints only one row. When `print_one_row` returns after being called the first time, `i` will have the value 11. The `for` statement in `print_all_rows` then increments `i` and tests whether it's less than or equal to 10. It's not, so the loop terminates and the function returns.
