will stand up. Consider this fragment, which computes the effective weight of an airplane, based on its true weight, length, and wingspan.

```
IF LENGTH >= 30 & LENGTH <= 50 THEN

   IF WING < .6*LENGTH THEN WEIGHT1 =
     (1+.08-.037)*WEIGHT;

   ELSE WEIGHT1=(1+.08+.045)*WEIGHT;
ELSE  IF LENGTH > 50 & LENGTH < 60 THEN

         IF WING < .6*LENGTH THEN
            WEIGHT1=(1+.09-.037)*
            WEIGHT;

          ELSE WEIGHT1=(1+.09+.045)*
            WEIGHT;

       ELSE  IF LENGTH > 60 & LENGTH
                < 80 THEN

        IF    WING < .6*LENGTH THEN
                WEIGHT1=(1+.105-.037)*
                WEIGHT;

        ELSE WEIGHT1=(1+.105+.045)*
                WEIGHT;

             ELSE IF WING < .6*LENGTH
                     THEN

                  WEIGHT1=(1+.122-
                   .037)*WEIGHT;
                  ELSE WEIGHT1=(1+
                   .122+.045)*WEIGHT;
```

When a program is well-structured, its layout can be made clean. For instance, programs that avoid labels and undisciplined branches should use indentation to emphasize the logical structure. (This program was originally displayed with vertical bars joining IF's with their corresponding ELSE's.) But indentation is no substitute for organization; tasteful formatting and top-to-bottom flow is no guarantee that the code cannot be improved.

Look at all the repetitions. The entire structure is turned inside out. If the test on WING is done first and the result saved for later use, and if we remove all the redundant tests, the code simplifies remarkably. Rearrangement also reveals the oversight in the original: the case where LENGTH is exactly 60 has been lumped in with the case for 80 and larger.