```
      IF(NUM.LT.0000000)GO TO 500
      IF(NUM.GT.9999999)GO TO 500
      IF(AMON.LT.00000000)GO TO 500
      IF(AMON.GT.99999999)GO TO 500
      IF(ITEM.LT.0000)GO TO 500
      IF(ITEM.GT.9999)GO TO 500
      GO TO 150
  500 WRITE(6,80)NUM,CUST,AMON,ITEM,IMM,IDD,IYY
      GO TO 150
```

Leaving aside the redundant zeros (after all, zero is zero, so adding more digits won't make it more precise), there is a suspicious regularity to the code: everything heads for statement 500.

Combining the logical conditions gives us the following version:

```
      IF (NUM .LT. 0 .OR. NUM .GT. 9999999
     $   .OR. AMON .LT. 0 .OR. AMON .GT. 99999999
     $   .OR. ITEM .LT. 0 .OR. ITEM .GT. 9999)
     $      WRITE(6,80) NUM, CUST, AMON, ITEM, IMM, IDD, IYY
      GO TO 150
```

This is still quite a mouthful, but since each part of the test has the same structure, and the parts are all combined with the same operator, it can be readily understood.

It is simpler to write good logical expressions in PL/I, but that is no guarantee that all expressions will be written as clearly as they can be:

```
IF K=0 | (¬(PRINT='YES' | PRINT='NO')) THEN DO;
```

The inversion and double parentheses slow comprehension. It seems better to distribute the "not" operation through the parenthesized expression. De Morgan's rules

```
¬(A | B)  <=>  ¬A & ¬B
¬(A & B)  <=>  ¬A | ¬B
```

tell us how:

```
IF K = 0 | (PRINT ¬= 'YES' & PRINT ¬= 'NO') THEN DO;
```

The expression is still not simple, but it is now in a form that more closely resembles how we speak. Note that we elected to keep the parentheses, even though none are necessary here, to make the operator binding unambiguous to the reader as well as the compiler.

A useful way to decide if some piece of code is clear or not is the "telephone test." If someone could understand your code when read aloud over the telephone, it's clear enough. If not, then it needs rewriting.

Use the "telephone test" for readability.

Judicious use of De Morgan's rules often improves the readability of programs by simplifying logical expressions. But care should be exercised in how they are applied. An example of the pitfalls of inverting logic comes from this routine to access a sparse matrix stored as a linear table. The function is supposed to return a