```
inword = NO
nword = 0
nsent = 0
WHILE (readch(char) = YES)
   IF (char = blank)
      inword = NO
   ELSE IF (char = period)
      nsent = nsent + 1
   ELSE IF (inword = NO)
      inword = YES
      nword = nword + 1
IF (nsent > 0)
   print nword/nsent
```

This is just about right to verify that the underlying algorithm works, even for fairly perverse sequences of blanks, words and periods. Now we can translate it into Fortran.

```
      INTEGER READCH, BLANK, PERIOD, CHAR, YES
      DATA BLANK /' '/, PERIOD /'.'/, NO /0/, YES /1/
      NWORD = 0
      NSENT = 0
      INWORD = NO
10    IF (READCH(CHAR) .EQ. NO) GOTO 90
         IF (CHAR .EQ. BLANK) INWORD = NO
         IF (CHAR .EQ. PERIOD) NSENT = NSENT + 1
         IF (CHAR .EQ. BLANK .OR. CHAR .EQ. PERIOD) GOTO 20
         IF (INWORD .EQ. YES) GOTO 20
            INWORD = YES
            NWORD = NWORD + 1
20       GOTO 10
90    IF (NSENT .GT. 0) AVG = FLOAT(NWORD) / FLOAT(NSENT)
      IF (NSENT .LE. 0) AVG = 0.0
      WRITE(6,91) AVG
91       FORMAT('0', 'AVERAGE WORDS PER SENTENCE =', F10.2)
      STOP
      END
```

Now the only remaining detail is READCH. Although there are several ways to write it, they all share the same basic approach — read in a whole card, then dole out the characters one at a time, reading a new card whenever the current one is exhausted. As a matter of timing, it's easier to read a new card only when it is really needed, not when the old one has just run out. Also, since READCH really returns two things, we return the character in the argument, and the end of file signal as the function value. This approach seems most convenient for Fortran, and eliminates the problem of choosing an end of file signal that is not a valid character.