It seems natural to assume that if J equals N+1, the new element will simply be added at the end. This is also convenient for entering the first element in an empty array, for which N is zero.

But the Fortran DO loop does us in. The loop is done once even if J exceeds N, so when N is zero, V(0) is accessed. The best solution is to protect the loop with an IF:

```
      IF (J .GT. N) GOTO 20
         DO 10 I = J, N
              K = N + J - I
              V(K + 1) = V(K)
10            CONTINUE
20    V(J) = VALUE
      N = N + 1
```

The PL/I DO loop behaves more suitably; it guarantees that if the termination condition is already met when the DO is begun, the body of the loop is executed zero times. Precautions like the following are unnecessary:

```
J=1; IF F1=1 THEN GO TO L2;
 L1: DO I=1 TO F1-1; PUT SKIP LIST(I,PRICE(I)); END;
 L2: DO I=F1 TO F2; TEMP=ROUND(PRICE(I)*(1-.01*DISCNT(J)),2);
     PUT SKIP LIST(I,TEMP);  J=J+1;
     END;
IF F2=32 THEN GO TO OTL;
 L3: DO I=F2+1 TO 32; PUT SKIP LIST(I,PRICE(I)); END;
GO TO OTL;
```

The extra tests and branches add no safety factor. Quite the contrary, their existence makes the program that much harder to read and understand. The code will perform identically if the redundant tests are eliminated.

```
J = 1;
DO I = 1 TO F1-1;
   PUT SKIP LIST(I, PRICE(I));
END;
DO I = F1 TO F2;
   PUT SKIP LIST(I, ROUND(PRICE(I)*(1 - 0.01*DISCNT(J)),2));
   J = J + 1;
END;
DO I = F2+1 TO 32;
   PUT SKIP LIST(I, PRICE(I));
END;
GOTO OTL;
```

The revised version appears bigger only because we have used white space more generously; it actually has fewer statements.

How can a conscientious programmer avoid errors like those we have shown? How can code be tested to exterminate those that do creep in? Many of our examples have illustrated what might be called "boundary-condition" errors, errors that arise at a critical data value or decision region. Things go wrong only there, not for the vast majority of cases "in the middle."

So it seems likely that one good strategy, both for writing and for testing, is to concentrate on the boundaries inherent in the program. For example, in the binary search above, we might guess that since the search is based on powers of two,