```
        LOOKU:IF MAZE(X-1,Y)='1'B THEN DO;
              BRANCH=LOOKR;          X=X-1;
                  GO TO SET;
                                END;

        LOOKL:IF MAZE(X,Y-1)='1'B THEN DO ;
              BRANCH=LOOKU;          Y=Y-1;
                  GO TO SET;
                                   END;

        LOOKD:IF MAZE(X+1,Y)='1'B THEN DO;
              BRANCH=LOOKL;          X=X+1;
                  GO TO SET;
                                END;

        LOOKR:IF MAZE(X,Y+1)='1'B THEN DO;
              BRANCH=LOOKD;          Y=Y+1;
                  GO TO SET;
                                END;
                                ELSE GO TO LOOKU;

             SET:I=I+1;

             POSITIONX(I)=X;          POSITIONY(I)=Y;

        IF X<N&X>1&Y<M&Y>1 THEN GO TO BRANCH;

        IF X=POSITIONX(1)&Y=POSITIONY(1) THEN DO;
                  IF IN2=0 THEN GO TO RUNUD;
                           ELSE GO TO RUNLR;
                                      END;

/*  NOW PICK OUT THOSE PARTS OF THE PATH FOLLOWED WHICH WENT IN A LOOP OR
    DEAD ENDED   */

/* NOTE THE TRANSFER STATEMENT WHICH CAUSES A TRANSFER TO WHAT LOOKS
   LIKE THE NEXT STATEMENT  -  ACTUALLY THE COMPILER CREATES A DUMMY STATEMENT
   BETWEEN THESE TWO TO END THE INNER DO LOOP  -  HENCE  THE PROGRAM MUST
   BRANCH AROUND IT   */

    SORT:DO J=I TO 2 BY-1;
         DO K=J-1 TO 1 BY -1;
             IF POSITIONX(K)=POSITIONX(J)&POSITIONY(K)=POSITIONY(J)
                       THEN DO;          II=II+1;       POINT(1,II)=J;
                            POINT(2,II)=K;     J=K+1;      GO TO ENDSORT;
    ENDSORT:END SORT;

/*  FINALLY SET UP THE FINAL MAZE WITH THE PATH FOLLOWED   */

    MERGE:DO KK=1 TO I;
             DO IK=1 TO II;
                  IF KK=POINT(2,IK) THEN KK=POINT(1,IK);
             END;

           XMAZE(POSITIONX(KK),POSITIONY(KK))=' ';
    END MERGE;
   PUT PAGE EDIT(((XMAZE(I,J) DO J=1 TO N) DO I=1 TO M))(LINE(33-M/2),
           (M) (COLUMN(40-N/2),(N) A(1),SKIP));

END EX510;
```

The interesting thing about this program is that it successfully ran a test case, despite all the errors we are about to unearth. To debug this code by running test cases alone would clearly take a long time. Just proofreading is hard enough,