Creating a Chess Player Part 3: Chess 0.5 (continued)

Larry R Atkin
Health Information Services
542 Michigan Av
Evanston IL 60202

Peter W Frey
Dept of Psychology
Northwestern University
Evanston IL 60201

Listing 1: The second half of Chess 0.5, written in Pascal. This portion of the program covers evaluation of terminal nodes, the look-ahead procedure and user commands (listing 1 continued on page 146).

```
PROCEDURE EVALU8;                          (* EVALUATE CURRENT POSITION *)

VAR
  INTV : TV;                               (* SCORE *)


  FUNCTION EVKING                          (* EVALUATE KING *)
    (A:RS;                                 (* KING BIT BOARD *)
     B:RS):TV;                             (* FRIENDLY PAWN BIT BOARD *)

  VAR
    INTS : TS;                             (* SCRATCH *)
    INRS : RS;                             (* SCRATCH *)
    INTV : TV;                             (* SCRATCH *)

  BEGIN
    ANDRS(INRS,A,CORNR);
    IF NULRS(INRS) THEN                    (* KING NOT IN CORNER *)
      INTV := 0
    ELSE
      INTV := FKSANQ;                      (* KING SAFELY IN CORNER *)

    INRS := A;
    IF NXTTS(INRS,INTS) THEN
    BEGIN
      ANDRS(INRS,ATKFR[INTS],B);           (* FIND PAWNS NEXT TO KING *)
      INTV := INTV + CNTRS(INRS)*FKPSHD;
                                           (* CREDIT EACH CLOSE PAWN *)
    END;

    EVKING := INTV;                        (* RETURN KING SCORE *)
  END;  (* EVKING *)


  FUNCTION EVMOBL                          (* EVALUATE MOBILITY *)
    (A,B:TP):TV;                           (* PIECE TYPES TO EVALUATE *)

  VAR
    INRS : RS;                             (* SCRATCH *)
    INTS : TS;                             (* SCRATCH *)
    INTV : TV;                             (* SCRATCH *)

  BEGIN
    IORRS(INRS,TPLOC[A],TPLOC[B]);         (* MERGE PIECE TYPES *)
    INTV := 0;                             (* INITIALIZE COUNT *)
    WHILE NXTTS(INRS,INTS) DO              (* COUNT ATTACKS *)
      INTV := INTV + CNTRS(ATKFR[INTS]);
    EVMOBL := INTV;                        (* RETURN TOTAL ATTACKS *)
  END;  (* EVMOBL *)


  FUNCTION EVPAWN                          (* EVALUATE PAWNS *)
    (A:RS;                                 (* LOCATION OF PAWNS *)
     B:TE;                                 (* PAWN FORWARD DIRECTION *)
     C:TR):TV;                             (* PAWN HOME RANK *)

  VAR
    INRS : RS;                             (* SCRATCH *)
    IMRS : RS;                             (* SCRATCH *)
    INTS : TS;                             (* SCRATCH *)
    INTV : TV;                             (* SCRATCH *)

  BEGIN
    SFTRS(INRS,A,S1);
    ANDRS(INRS,INRS,A);                    (* BIT SET FOR SIDE BY SIDE *)
    INTV := CNTRS(INRS)*FPFLNX;            (* SCORE PHALANX *)

    SFTRS(INRS,A,B1);
    ANDRS(INRS,INRS,A);                    (* BIT SET FOR PAWN DEFENSE *)
    INTV := INTV + CNTRS(INRS)*FPCONN;     (* CREDIT CONNECTED PAWNS *)

    SFTRS(INRS,A,B2);
    ANDRS(INRS,INRS,A);
    INTV := INTV + CNTRS(INRS)*FPCONN;     (* AND OTHER CONNECTED PAWNS *)

    SFTRS(INRS,A,B);                       (* MOVE FORWARD *)
    NOTRS(INRS,TPLOC[MT]);                 (* OCCUPIED SQUARES *)
    ANDRS(INRS,INRS,IMRS);                 (* BLOCKED PAWNS *)
    INTV := INTV - CNTRS(INRS)*FPBLOK;     (* PENALIZE BLOCKED PAWNS *)

    CPYRS(INRS,A);
    WHILE NXTTS(INRS,INTS) DO              (* FOR EACH PAWN *)
      INTV := INTV +(ABS(ORD(C)-ORD(XTSR[INTS])))*FPADCR[XTSF[INTS]];
                                           (* CREDIT PAWN ADVANCEMENT *)

    EVPAWN := INTV;                        (* RETURN PAWN SCORE *)
  END;  (* EVPAWN *)
```

This month we conclude the listing and commentary of Chess 0.5 begun last issue. The program was written by Larry Atkin, who is coauthor with David Slate of the world championship chess program, Chess 4.6. The program is readily adaptable to personal computers having Pascal systems such as the UCSD Pascal project software. Part 4 concludes the series with a discussion of chess strategy and tactics.

Evaluating Terminal Positions

Another important aspect of any chess program is the function which provides a static evaluation of terminal positions in the look-ahead tree. In the present program, this routine also doubles as a preliminary scoring function for sorting moves at the first ply, at the beginning of the look-ahead search. Since the evaluation function is used repetitively in the search, efficiency demands that it be carefully engineered. We have left this task as an exercise for the reader. Our function presently includes only a few basic essentials.

The most important feature is material. We employ essentially the same function for this that is used by Chess 4.5. A trade-down bonus is also incorporated, ie: trade pieces but not pawns when ahead in material. A second feature which is considered is piece mobility. The mobility of Knights and Bishops is weighted more heavily than that for Roks and Queens. Special credit is given to a King which is located in one of the four corner squares in each corner of the board, ie: 16 squares total. This encourages early castling. Pawn structure is considered by providing a bonus for advancing the pawns in the four center files, for having a pawn near the King, and for having a pawn adjacent to or defended by another pawn. This indirectly penalizes isolated or backward pawns. There is a direct penalty if the square in front of a pawn is occupied. The position of the Rooks is considered by providing a bonus for placing a Rook on the seventh rank and for attacking another Rook of the same color (ie: doubled Rooks). The executive routine for these assessments is EVALU8.

The Look-Ahead Procedure

The look-ahead procedure is controlled by an executive routine called SEARCH. Several subprocedures are also defined which handle specific tasks. NEWBST keeps track of the move which is currently thought to be best, and dynamically reorders the moves at the first ply level each time a new best-move is selected. MINMAX determines whether the move under consideration will produce an α-β cutoff. SCOREM is called into action when the program can find no legal moves at a node. It determines whether the position should be scored as a checkmate or as a stalemate. SELECT is responsible for move ordering at each node. It determines whether there are any more moves to be searched and if so, makes sure that they are generated in the correct order (ie: captures, killers, castling moves, and then the remaining moves).

SEARCH incorporates a number of important features which make the look-ahead search more efficient. These include staged move generation, preliminary ordering scores, setting a narrow α-β window at the beginning of the search, conducting the search in an iterative fashion, and dynamically recording moves at the first ply as the search proceeds. Because of these features, the full-width search takes a long time instead of taking forever.

User Commands

For the user's convenience, the program should be able to respond to a few simple commands. Inputs to the program are processed by a lengthy routine, READER, which has many component subprocedures. The translation of the input string is handled by a group of routines: RDRERR, RDRGNT, RDRSFT, RDRCMP, RDLINE, RDRMOV and RDRNUM. Each of the commands is executed by a separate routine.

When the human player wishes to terminate the game before it has reached its conclusion (eg: when he is hopelessly lost and does not want to stay around to be crushed), he can simply type an END command and the ENDCMD routine will terminate the program. If the user simply wishes to start a new game, he can type INIT and the INICMD routine will set up for a new game.

If the user would like to set up a specific position from the previous game or some other game, he can call the BOACMD routine, which will set up any position he desires. To use this instruction, the pieces are designated in the standard way (eg: K, Q, R, B, N and P) and the colors are designated by L for light and D for dark. The board is described by starting at the lower lefthand corner and listing, row by row, the 64 squares. Numbers are used to represent consecutive empty squares. The command to set up the position after 1. P-K4, P-K4, 2. N-KB3, N-QB3 is: BOARD, LRNBQKB1 RPPPP1PPP5N24P34DP33N4PPPP1PPPR1B QKBNR.

If the human player is lazy or simply wishes to test the program, he or she can type GO and the machine will select a move. By repeatedly typing GO the user can sit back and watch the machine play against itself. The routine that handles this is GONCMD. To specify a value for selected program parameter variables, the player can use LETCMD. For example, the amount of time the machine spends calculating a move can be controlled by specifying a limit for the number of nodes to be searched. The command LET FNODEL = 1000 will cause the machine to set a target value of 1000 for the number of nodes to be searched. In this case it will not start another iteration if it has already searched 1000 nodes. If the user is confused about the current board configuration, the command PRINT will activate PRICMD which calls PRINTB for a representation (8 by 8 array) of the board. For diagnostic purposes the user can also ask for other information. The routine PAMCMD is activated by PB and provides an 8 by 8 attack map for each of the 64 squares. The routine POPCMD is activated by PO and gives information concerning the side to move (White or Black), the en passant status after the last move, the present castle status and the move number. If the user types PM, the routine PMVCMD will provide a list of all moves which are legal for the side to move in the current position. The command PL activates PLECMD which prints the value of a designated variable; for example, the user can determine the present limit for the number of nodes to be searched by typing PL FNODEL.

The user also has control over several switches. He can ask the machine to repeat (echo) each entry, to pause after 20 lines of output, and to reply automatically each time the opponent enters a move. These switches are set by the switch commands (eg: SW EC OFF), and are processed by SWICMD. If the user wishes to manually alter one or more of the status conditions (eg: side to move, move number, en passant, castling), this can be done by activating STACMD.

Notes on Notation

The program also processes standard chess notation. This is not strictly necessary. Many programs use their own convention for entering and reporting moves. A common procedure is to denote the squares using a number (1 through 8) for each row and a letter (A through H) for each column. A move is defined by listing the present square of the piece and then the destination square. For example, the common opening move, P-K4, would be E2E4. Moving the White Knight on the kingside from its original square to KB3 would be G1F3. This convention works nicely but it forces an experienced chess player to learn a new system. Most would prefer standard chess notation.

Because there are multiple ways to express the same move in standard notation, the translation routine needs to be fairly sophisticated. Consider a position in which the White Queen's Rook is on its original square and the neighboring Knight and Bishop have been moved. A move which places the Rook on the Queen Bishop file can be designated as R-B1, R-QB1, R/1-B1, R/1-QB1, R/R1-B1, or R/R1-QB1. It is important that the program recognize that each of these character strings represents the same move. How is this done?

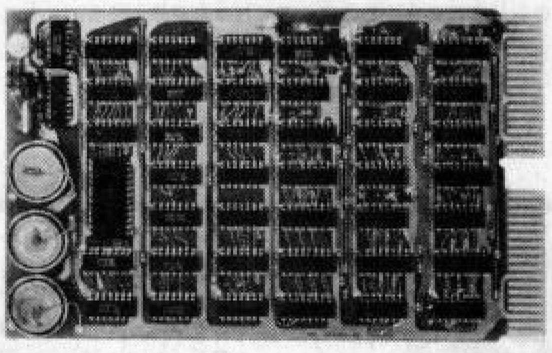
One way is to have the machine generate a list of all legal moves and then compare each of these with the move entered by the player. If his move matches one on the list, that move is noted. The rest of the list is then checked and if no more matches are found, the noted move is assumed to be the correct one. If no match is found, the machine prints "illegal move." If a second match is found (eg: P-B3 matches both P-KB3 and P-QB3), the machine prints "ambiguous move." The process of translating the opponent's move into machine compatible form and checking its legality or ambiguity is done by YRMOVE. The process of translating the machine's move into standard notation is handled by MYMOVE. Both of these procedures call MINENG, which is responsible for constructing the appropriate character strings.

Final Thoughts

This completes our listing of our demonstration chess program. Despite the program's length, there are many desirable features which have been omitted. The reader with an interest in chess and programming should use this listing as a starting point for developing a program. The time required for move calculation can be reduced by writing machine dependent code for some of the frequently used routines. There are also features which can be added to improve the level of play.

One useful addition would be an opening library. An effective technique for this is described by Slate and Atkin in their chapter in *Chess Skill in Man and Machine* (P W Frey, editor, Springer-Verlag, New York, 1977). An opening library provides the user with a challenging set of opening moves and directs the game into situations which are familiar to the experienced chess player. By including various options at the early choice points and using a random selection procedure, the programmer can insure that the machine will not always select the same move sequence. The programmer can also give the user the option of specifying a particular opening against which he would like to practice. For important matches, the programmer can prepare surprise openings for the machine in order to gain a psychological edge on the opponent.

Text continued on page 157



Listing 1, continued from page 140:

```
  FUNCTION EVROOK                            (* EVALUATE ROOKS *)
    (A:RS;                                   (* ROOK LOCATIONS *)
     B:RS):TV;                               (* SEVENTH RANK *)

  VAR
    INTV : TV;                               (* SCRATCH *)
    INTI : TI;                               (* SCRATCH *)
    INTS : TS;                               (* SCRATCH *)
    INRS : RS;                               (* SCRATCH *)

  BEGIN
    INTV := 0;                               (* INITIALIZE *)
    INRS := A;
    IF NXTTS(INRS,INTS) THEN                 (* LOCATE FIRST ROOK *)
    BEGIN
      ANDRS(INRS,A,ATKFR[INTS]);
      IF NOT NULRS(INRS) THEN                (* ROOK ATTACKS FRIENDLY ROOK *)
        INTV := INTV + FROUBL;               (* GIVE DOUBLED ROOK CREDIT *)
    END;

    ANDRS(INRS,A,B);                         (* ROOKS ON SEVENTH *)
    INTI := CNTRS(INRS);
    EVROOK := INTV + INTI*INTI*FRK7TH;  (* CREDIT ROOKS ON SEVENTH *)
  END;  (* EVROOK *)



BEGIN
  IF XTMV[JNTM]*MBVAL[JNTK] + MAXPS <= BSTVL[JNTK-2] THEN
                                             (* MOVE WILL PRUNE ANYWAY *)
    INTV := XTMV[JNTM] * MBVAL[JNTK]
  ELSE
  BEGIN
    INTV :=(   FWPAWN*(EVPAWN(TPLOC[LP],S2,R2)-EVPAWN(TPLOC[DP],S4,R7))
             + FWMINM*(EVMOBL(LB,LN)           -EVMOBL(DB,DN)          )
             + FWMAJM*(EVMOBL(LR,LQ)           -EVMOBL(DR,DQ)          )
             + FWROOK*(EVROOK(TPLOC[LR],XRRS[R7])
                      -EVROOK(TPLOC[DR],XRRS[R2])                      )
             + FWKING*(EVKING(TPLOC[LK],TPLOC[LP])
                      -EVKING(TPLOC[DK],TPLOC[DP])                     )
            ) DIV 64;
    MAXPS := MAX(MAXPS,ABS(INTV));
    INTV := XTMV[JNTM]*(MBVAL[JNTK]+INTV);
  END;
  IF SWTR THEN
  BEGIN
    WRITE(" EVALU8",JNTK,JNTW,INDEX[JNTK],INTV);
    PRIMOV(MOVES[INDEX[JNTK]]);
  END;
  VALUE[INDEX[JNTK]] := INTV;                (* RETURN SCORE *)
END;  (* EVALU8 *)


FUNCTION SEARCH                              (* SEARCH LOOK-AHEAD TREE *)
  :TW;                                       (* RETURNS THE BEST MOVE *)

LABEL
  11,                                        (* START NEW PLY *)
  12,                                        (* TRY DIFFERENT FIRST MOVE *)
  13,                                        (* FLOAT VALUE BACK UP *)
  14,                                        (* FIND ANOTHER MOVE *)
  15,                                        (* BACK UP A PLY *)
  16;                                        (* EXIT SEARCH *)


  PROCEDURE NEWBST                           (* SAVE BEST MOVE INFORMATION *)
    (A:TK);                                  (* PLY OF BEST MOVE *)

  VAR
    INTW : TW;                               (* MOVES INDEX *)
    INRM : RM;                               (* SCRATCH *)

  BEGIN
    BSTMV[A] := INDEX[A+1];                  (* SAVE BEST MOVE *)
    IF A = AK THEN                           (* AT FIRST PLY *)
    BEGIN
      INRM := MOVES[BSTMV[A]];               (* SAVE BEST MOVE *)
      FOR INTW := BSTMV[A]-1 DOWNTO AW+1 DO
        MOVES[INTW+1] := MOVES[INTW];        (* MOVE OTHER MOVES DOWN *)
      MOVES[AW+1] := INRM;                   (* PUT BEST AT BEGINNING *)
      BSTMV[AK] := AW+1;                     (* POINTS TO BEST MOVE *)
    END
    ELSE
      IF NOT MOVES[BSTMV[A]].RMCA THEN
        KILLR[JNTK] := MOVES[BSTMV[A]]; (* SAVE KILLER MOVE *)
  END;  (* NEWBST *)


  FUNCTION MINMAX                            (* PERFORM MINIMAX OPERATION *)
    (A:TK)                                   (* PLY TO MINIMAX AT *)
     :TB;                                    (* TRUE IF REFUTATION *)

  BEGIN
    MINMAX := FALSE;                         (* DEFAULT IS NO PRUNING *)
    IF SWTR THEN
      WRITE(" MINMAX",A,-BSTVL[A-1],BSTVL[A],-BSTVL[A+1]);
    IF -BSTVL[A+1] > BSTVL[A] THEN
    BEGIN
      BSTVL[A] := -BSTVL[A+1];
      NEWBST(A);                             (* SAVE BEST MOVE *)
      MINMAX := BSTVL[A+1] <= BSTVL[A-1];
                                             (* RETURN TRUE IF REFUTATION *)
      IF SWTR THEN
        WRITE(" NEW BEST. PRUNE: ",BSTVL[A+1] <= BSTVL[A-1]);
    END;
    IF SWTR THEN
      WRITELN;                               (* PRINT TRACE LINE *)
  END;  (* MINMAX *)
```

```
  PROCEDURE SCOREM;                          (* SCORE MATE *)

  BEGIN
    MOVES[INDEX[JNTK]].RMMT := TRUE;         (* INDICATE MATE *)
    IF MOVES[INDEX[JNTK]].RMCH THEN          (* CHECKMATE *)
      VALUE[INDEX[JNTK]] :=  64*JNTK - ZV
    ELSE                                     (* STALEMATE *)
      VALUE[INDEX[JNTK]] := 0;
    IF SWTR THEN
      WRITELN(" SCOREM",JNTK,JNTW,INDEX[JNTK],VALUE[INDEX[JNTK]]);
  END;  (* SCOREM *)


  FUNCTION SELECT                            (* SELECT NEXT MOVE TO SEARCH *)
     :TB;                                    (* TRUE IF MOVE RETURNED *)

  LABEL
    21,                                      (* NEW SEARCH MODE *)
    22;                                      (* EXIT SELECT *)

  VAR
    INTB : TB;                               (* RETURN VALUE *)
    INTK : TK;                               (* SCRATCH *)
    INTW : TW;                               (* MOVE INDEX *)
    INTW : TW;                               (* SCRATCH *)
    INTV : TV;                               (* SCRATCH *)


    PROCEDURE SELDON;                        (* SELECT EXIT - DONE.
                                                CALLED WHEN NO FURTHER
                                                MOVES ARE TO BE SEARCHED
                                                FROM THIS POSITION.
                                                THE CURRENT POSITION MUST
                                                HAVE BEEN EVALUATED. *)

    BEGIN
      INTB := FALSE;                         (* RETURN NO MOVE SELECTED *)
      IF SWTR THEN
        WRITELN(" SELECT",JNTK," END.");
      GOTO 22;                               (* EXIT SELECT *)
    END;  (* SELDON *)



    PROCEDURE SELMOV                         (* SELECT EXIT - SEARCH.
                                                CALLED WHEN A MOVE TO
                                                BE SEARCHED HAS BEEN
                                                FOUND. *)
      (A:TW);                                (* INDEX TO SELECTED MOVE *)

    BEGIN
      INTB := TRUE;                          (* RETURN MOVE SELECTED *)
      INDEX[JNTK+1] := A;                    (* POINT TO SELECTED MOVE *)
      MOVES[A].RMSU := TRUE;                 (* FLAG MOVE AS SEARCHED *)
      IF SWTR THEN
      BEGIN
        WRITE(" SELECT",JNTK,ORD(SRCHM[JNTK]),A);
        PRIMOV(MOVES[A]);
      END;
      GOTO 22;                               (* EXIT SELECT *)
    END;  (* SELMOV *)


    PROCEDURE SELNXT                         (* SELECT EXIT - NEW MODE.
                                                CALLED WHEN A NEW SEARCH
                                                MODE IS TO BE SELECTED *)
      (A:TH);                                (* NEW SEARCH MODE *)

    BEGIN
      INDEX[JNTK+1] := LINDX[JNTK]-1;        (* RESET MOVES POINTER *)
      SRCHM[JNTK] := A;                      (* CHANGE SEARCH MODE *)
      GOTO 21;                               (* EXECUTE NEXT MODE *)
    END;  (* SELNXT *)



    PROCEDURE SELANY;                        (* SEARCH ALREADY GENERATED
                                                AND NOT ALREADY SEARCHED *)

    VAR
      INTW : TW;                             (* MOVES INDEX *)

    BEGIN
      FOR INTW := INDEX[JNTK+1]+1 TO JNTW-1 DO
        IF NOT MOVES[INTW].RMSU THEN
          SELMOV(INTW);
    END;  (* SELANY *)


  BEGIN
  21:  (* NEW SEARCH MODE *)
    CASE SRCHM[JNTK] OF

      H0:  (* INITIALIZE FOR NEW MOVE *)
           BEGIN
             MVSEL[JNTK] := 0;               (* CLEAR MOVES SEARCHED *)
             INTV := BSTVL[JNTK-2];          (* SAVE ALPHA *)
             BSTVL[JNTK-2] := -ZV;           (* INHIBIT PRUNING IN EVALU8 *)
             MAXPS := 0;                     (* INITIALIZE MAXIMUM POSITIONAL
                                                SCORE *)
             GENALL;                         (* GENERATE ALL MOVES *)
             FOR INTW := AW+1 TO JNTW-1 DO
             BEGIN
               IF UPDATE(MOVES[INTW]) THEN
               BEGIN
                 INDEX[JNTK] := INTW;        (* POINT TO CURRENT MOVE *)
                 EVALU8                      (* SCORE POSITION *)
               END;
               DNDATE(MOVES[INTW]);
             END;
             BSTVL[JNTK-2] := INTV;          (* RESTORE ALPHA *)
             SORTIT(VALUE,MOVES,JNTW-1);
                                             (* SORT PRELIMINARY SCORES *)
             FOR INTK := AK TO ZK DO
               KILLR[INTK] := NULMV;         (* CLEAR KILLER TABLE *)
```

Listing 1, continued:

```
       IF SWTR OR SWPS THEN
         FOR INTW := AW+1 TO JNTW-1 DO
         BEGIN
           WRITE(" PRELIM",INTW,VALUE[INTW]);
           PRIMOV(MOVES[INTW]);  (* PRINT PRELIMINARY SCORES *)
           IF INTW/LPP = INTW DIV LPP THEN
             PAUSER;
         END;
       SELNXT(H6);                  (* SEARCH ALL MOVES *)
     END;

H1:  (* INITIALIZE AT NEW DEPTH *)
     BEGIN
       MVSEL[JNTK] := 0;            (* CLEAR MOVES SEARCHED *)
       IF JNTK > JMTK THEN
       BEGIN
         EVALU8;                    (* EVALUATE CURRENT POSITION *)
         INDEX[JNTK+1] := AW;
         BSTVL[JNTK+1] := -VALUE[INDEX[JNTK]];
         IF MINMAX(JNTK) OR (JNTK = ZK) THEN
           SELDON;                  (* THIS MOVE PRUNES *)
         SRCHM[JNTK] := H2;         (* CAPTURE SEARCH *)
       END
       ELSE
         SRCHM[JNTK] := H3;         (* CAPTURES IN FULL SEARCH *)
       GENCAP;                      (* GENERATE CAPTURES *)
       SELNXT(SRCHM[JNTK]);         (* CHANGE SEARCH MODE *)
     END;

H2:  (* CAPTURE SEARCH *)
     BEGIN
       INTW := AW;                  (* BEST MOVE POINTER *)
       INTV := AV;                  (* BEST VALUE *)
       FOR IMTW := LINDX[JNTK] TO JNTW-1 DO
         WITH MOVES[IMTW] DO
           IF NOT RMSU THEN
             IF ABS(XTPV[RMCP]) > INTV THEN
             BEGIN
               INTV := ABS(XTPV[RMCP]);

               INTW := IMTW;
             END;
       IF INTW <> AW THEN           (* MOVE FOUND *)
         SELMOV(INTW)               (* SELECT BIGGEST CAPTURE *)
       ELSE
         SELDON;                    (* QUIT *)
     END;

H3:  (* FULL WIDTH SEARCH - CAPTURES *)
     BEGIN
       INTW := AW;                  (* BEST MOVE POINTER *)
       INTV := AV;                  (* BEST VALUE *)
       FOR IMTW := LINDX[JNTK] TO JNTW-1 DO
         WITH MOVES[IMTW] DO
           IF NOT RMSU THEN
             IF ABS(XTPV[RMCP]) > INTV THEN
             BEGIN
               INTV := ABS(XTPV[RMCP]);

               INTW := IMTW;
             END;
       IF INTW <> AW THEN           (* MOVE FOUND *)
         SELMOV(INTW)               (* SELECT BIGGEST CAPTURE *)
       ELSE
         IF NOT NULMVB(KILLR[JNTK]) THEN
         BEGIN
           IMTW := JNTW;            (* SAVE CURRENT MOVES INDEX *)
           GENFSL(XRSS[KILLR[JNTK].RMFR]);
                                    (* GENERATE MOVE BY KILLER *)
           SRCHM[JNTK] := H4;       (* SET NEXT SEARCH MODE *)
           FOR INTW := IMTW TO JNTW-1 DO
                                    (* LOOK AT MOVES BY KILLER *)
             IF KILLR[JNTK].RMTO = MOVES[INTW].RMTO THEN
               SELMOV(INTW);        (* SELECT KILLER MOVE *)
         END;
       SELNXT(H4);                  (* GO TO NEXT STATE *)
     END;

H4:  (* INITIALIZE SCAN OF CASTLE MOVES AND OTHER MOVES
       BY KILLER PIECE *)
     BEGIN
       GENCAS;                      (* GENERATE CASTLE MOVES *)
       SELNXT(H5);                  (* GO TO NEXT STATE *)
     END;

H5:  (* FULL WIDTH SEARCH - CASTLES AND OTHER MOVES BY KILLER
       PIECE *)
     BEGIN
       SELANY;                      (* SELECT ANY MOVE *)
       GENFSL(ALLOC[JNTK]);         (* GENERATE REMAINING MOVES *)
       SELNXT(H6);                  (* NEXT SEARCH MODE *)
     END;

H6:  (* FULL WIDTH SEARCH - REMAINING MOVES *)
     BEGIN
       SELANY;                      (* SELECT ANYTHING ON LIST *)
       IF MVSEL[JNTK] = 0 THEN
         SCOREM;                    (* SCORE MATE *)
       SELDON;                      (* EXIT SELECT *)
     END;

H7:  (* RESEARCH FIRST PLY *)
     BEGIN
       JNTW := LINDX[AK+1];         (* POINT TO ALREADY GENERATED
                                       MOVES *)
       MVSEL[AK] := 0;              (* RESET MOVES SEARCHED *)
       FOR INTW := AW+1 TO JNTW-1 DO
         MOVES[INTW].RMSU := FALSE;
                                    (* CLEAR SEARCHED BIT *)
       IF SWTR THEN
         WRITELN(" REDO  ",JMTK,BSTVL[AK-2],BSTVL[AK-1]);
       SELNXT(H6);                  (* SEARCH ALL MOVES *)
     END;
END;
```

```
  22:  (* SELECT EXIT *)
    SELECT := INT0;                     (* RETURN VALUE *)
  END;  (* SELECT *)

BEGIN  (* SEARCH *)
  BSTMV[AK] := AW;                      (* INITIALIZE MOVE *)
  INDEX[JNTK] := AW;                    (* INITIALIZE TREE *)
  MOVES[AW] := LSTMV;                   (* INITIALIZE MOVE *)
  EVALU8;                               (* INITIAL GUESS AT SCORE *)
  BSTVL[AK-2] := VALUE[AW] - WINDOW;    (* INITIALIZE ALPHA-BETA
                                           WINDOW *)
  BSTVL[AK-1] := - VALUE[AW] - WINDOW;
  JMTK := AK+1;                         (* INITIALIZE ITERATION NUMBER *)
  WHILE (NODES < FNODEL) AND (JMTK < MAX(ZK DIV 2, ZK-8)) DO
  BEGIN

11:  (* START NEW PLY *)
    BSTVL[JNTK] := BSTVL[JNTK-2];       (* INITIALIZE ALPHA *)

12:  (* DIFFERENT FIRST MOVE *)
    IF NOT SELECT THEN
    BEGIN
      BSTVL[JNTK] := VALUE[INDEX[JNTK]];
      NEWBST(JNTK);
    END
    ELSE
    BEGIN
      IF UPDATE(MOVES[INDEX[JNTK+1]]) THEN
        GOTO 11                         (* START NEW PLY *)
      ELSE
      BEGIN
        DNDATE(MOVES[INDEX[JNTK]]);
        GOTO 12;                        (* FIND ANOTHER MOVE *)
      END;

13:  (* FLOAT VALUE BACK *)
      IF MINMAX(JNTK) THEN
        GOTO 15;                        (* PRUNE *)

14:  (* FIND ANOTHER MOVE AT THIS PLY *)
      IF SELECT THEN
        IF UPDATE(MOVES[INDEX[JNTK+1]]) THEN
          GOTO 11                       (* START NEW PLY *)
        ELSE
        BEGIN
          DNDATE(MOVES[INDEX[JNTK]]);
          GOTO 14;                      (* FIND ANOTHER MOVE *)
        END;
    END;

15:  (* BACK UP A PLY *)
    IF JNTK > AK THEN
    BEGIN (* NOT DONE WITH ITERATION *)
      DNDATE(MOVES[INDEX[JNTK]]);       (* RETRACT MOVE *)
      GOTO 13;
    END;

     (* DONE WITH ITERATION *)
    IF (BSTVL[AK] <= BSTVL[AK-2]) OR (BSTVL[AK] >= -BSTVL[AK-1]) THEN
    BEGIN  (* NO MOVE FOUND *)
      IF MVSEL[AK] = 0 THEN
      BEGIN  (* NO LEGAL MOVES *)
        GOTO 16;                        (* GIVE UP *)
      END;
      BSTVL[AK-2] := -ZV;               (* SET ALPHA-BETA WINDOW LARGE *)
      BSTVL[AK-1] := -ZV;
      SRCHM[AK] := H7;
      JNTW := AK+1;
      GOTO 11;                          (* TRY AGAIN *)
    END;
    BSTVL[AK-2] := BSTVL[AK] - WINDOW;  (* SET ALPHA BETA WINDOW *)
    BSTVL[AK-1] := - BSTVL[AK] - WINDOW;
    JMTK := JMTK+1;                     (* ADVANCE ITERATION NUMBER *)
    SRCHM[AK] := H7;
  END;

  16:  (* EXIT SEARCH *)
  SEARCH := BSTMV[AK];                  (* RETURN BEST MOVE *)
END;  (* SEARCH *)



PROCEDURE READER;                       (* READ INPUT FROM USER *)

LABEL
  11;                                   (* COMMAND FINISHED EXIT *)

VAR
  INRA : RA;                            (* SCRATCH TOKEN *)
  INTJ : TJ;                            (* ECHO COMMAND INDEX *)


  PROCEDURE RDRERR(A:RN);               (* PRINT DIAGNOSTIC AND EXIT *)

  VAR
    INTJ : TJ;                          (* STRING INDEX *)
    INTN : TN;                          (* MESSAGE INDEX *)

  BEGIN
    IF NOT SWEC THEN                    (* ECHO LINE IF NOT ALREADY
                                           DONE *)
    BEGIN
      WRITE(" ");
      FOR INTJ := AJ TO ZJ-1 DO
        WRITE(ILINE[INTJ]);             (* WRITE INPUT LINE *)
      WRITELN;
    END;
    FOR INTJ := AJ TO JNTJ DO
      WRITE(" ");                       (* LEADING BLANKS BEFORE ARROW *)
    WRITELN("^");                       (* POINTER TO ERROR *)
    FOR INTN := AN TO ZN DO
      WRITE(A[INTN]);                   (* WRITE DIAGNOSTIC *)
    WRITELN;
    GOTO 11;                            (* COMMAND EXIT *)
  END;  (* RDRERR *)
```

Listing 1, continued:

```
FUNCTION RDRGNT(VAR Æ:RA):TB;          (* GET NEXT TOKEN FROM COMMAND
                                          RETURNS TOKEN IN A.
                                          RETURNS TRUE IF NON-EMPTY
                                          TOKEN.
                                          A TOKEN IS ANY CONSECUTIVE
                                          COLLECTION OF ALPHANUMERIC
                                          CHARACTERS.
                                          LEADING SPECIAL CHARACTERS
                                          IGNORED. *)

VAR
  INTJ : TJ;                           (* STRING INDEX *)

BEGIN
  WHILE (JNTJ < ZJ) AND (ORD(ILINE[JNTJ]) >= ORD("+")) DO
    JNTJ := JNTJ+1;
  A := "          ";
  INTJ := AA;
  WHILE (JNTJ < ZJ) AND (INTJ < ZA) AND (ILINE[JNTJ] IN ["A".."9"]) DO
  BEGIN
    A[INTJ] := ILINE[JNTJ];            (* COPY CHARACTER TO TOKEN *)
    INTJ := INTJ+1;                    (* ADVANCE POINTERS *)
    JNTJ := JNTJ+1;
  END;
  RDRGNT := INTJ <> AA;                (* RETURN TRUE IF ANYTHING
                                          MOVED *)
  WHILE (INTJ < ZJ) AND (ILINE[JNTJ] IN ["A".."9"]) DO
    JNTJ := JNTJ+1;                    (* SKIP REST OF TOKEN *)
END;  (* RDRGNT *)


PROCEDURE RDRSFT;                      (* SKIP FIRST TOKEN IN COMMAND
                                          LINE *)

VAR
  INRA : RA;                           (* SCRATCH *)
  INTB : TB;                           (* SCRATCH *)

BEGIN
  JNTJ := AJ;                          (* INITIALIZE SCAN *)
  INTB := RDRGNT(INRA);                (* THROW AWAY FIRST TOKEN *)
END;  (* RDRSFT *)


PROCEDURE RDRCMD                       (* TEST FOR AND EXECUTE COMMAND
                                          EXITS TO COMMAND EXIT IF
                                          COMMAND IS PROCESSED. *)
```

```
  (A:RA;                               (* POTENTIAL COMMAND KEYWORD *)
   PROCEDURE XXXCMD);                  (* PROCEDURE TO EXECUTE
                                          COMMAND *)

BEGIN
  IF INRA = A THEN
  BEGIN
    XXXCMD;                            (* EXECUTE COMMAND *)
    GOTO 11;                           (* EXIT *)
  END;
END;  (* RDRCMD *)


PROCEDURE RDLINE;                      (* GET NEXT INPUT LINE FROM
                                          USER *)

VAR
  INTC : TC;                           (* SCRATCH *)
  INTJ : TJ;                           (* STRING INDEX *)

BEGIN
  READLN;                              (* ADVANCE TO NEXT LINE *)
  INTJ := AJ;
  WHILE NOT EOLN AND (INTJ < ZJ) DO
  BEGIN
    READ(ICARD[INTJ]);                 (* COPY INPUT LINE *)
    INTJ := INTJ+1;
  END;
  WHILE NOT EOLN DO
    READ(INTC);                        (* SKIP REST OF INPUT LINE *)
  WHILE INTJ < ZJ DO
  BEGIN
    ICARD[INTJ] := " ";                (* BLANK REST OF LINE *)
    INTJ := INTJ+1;
  END;
  ICARD[ZJ] := ";";                    (* SET END OF COMMAND *)
  JNTJ := AJ;                          (* RESET INPUT LINE POINTER *)
END;  (* RDLINE *)


FUNCTION RDRMOV:TB;                    (* EXTRACT NEXT COMMAND
                                          FROM INPUT LINE.
                                          RETURNS TRUE IF NON-EMPTY
                                          COMMAND. *)

VAR
  INTJ : TJ;                           (* STORING POINTER *)

BEGIN
  WHILE (JNTJ < ZJ) AND (ICARD[JNTJ] = " ") DO
    JNTJ := JNTJ+1;                    (* SKIP LEADING BLANKS *)
  INTJ := AJ;
  WHILE (JNTJ < ZJ) AND (ICARD[JNTJ] <> ";") DO
  BEGIN
    ILINE[INTJ] := ICARD[JNTJ];
    INTJ := INTJ+1;
    JNTJ := JNTJ+1;
  END;
  IF (ICARD[JNTJ] = ";") AND (JNTJ < ZJ) THEN
    JNTJ := JNTJ+1;                    (* SKIP SEMI-COLON *)
  RDRMOV := INTJ <> AJ;                (* RETURN TRUE IF NON-EMPTY *)
  WHILE INTJ < ZJ DO
  BEGIN
    ILINE[INTJ] := " ";                (* BLANK FILL LINE *)
    INTJ := INTJ+1;
  END;
  ILINE[ZJ] := ";";                    (* STORE COMMAND TERMINATOR *)
  JNTJ := AJ;                          (* PRESET COMMAND SCAN *)
END;  (* RDRMOV *)


FUNCTION RDRNUM:TI;                    (* CRACK NUMBER FROM COMMAND
                                          LINE.  RETURNS NUMBER IF NO
                                          ERROR.  EXITS TO COMMAND EXIT
                                          IF ERROR. *)

VAR
  INTB : TB;                           (* SIGN *)
  INTI : TI;                           (* VALUE *)

BEGIN
  WHILE (JNTJ < ZJ) AND (ILINE[JNTJ] = " ") DO
    JNTJ := JNTJ+1;                    (* SKIP LEADING BLANKS *)
  IF ILINE[JNTJ] = "-" THEN
  BEGIN
    INTB := TRUE;                      (* NUMBER IS NEGATIVE *)
    JNTJ := JNTJ+1;                    (* ADVANCE CHARACTER POINTER *)
  END
  ELSE
  BEGIN
    INTB := FALSE;                     (* NUMBER IS POSITIVE *)
    IF ILINE[JNTJ] = "+" THEN
      JNTJ := JNTJ+1;                  (* SKIP LEADING + *)
  END;
  INTI := 0;
  WHILE ILINE[JNTJ] IN ["0".."9"] DO
  BEGIN
    IF INTI < MAXINT/10 THEN
      INTI := 10*INTI+ORD(ILINE[JNTJ])-ORD("0")
    ELSE
      RDRERR(" NUMBER TOO LARGE              ");
    JNTJ := JNTJ+1;                    (* ADVANCE *)
  END;
  IF ILINE[JNTJ] IN ["A".."Z"] THEN
    RDRERR(" DIGIT EXPECTED                ");
  IF INTB THEN
    INTI := -INTI;                     (* COMPLEMENT IF NEGATIVE *)
  RDRNUM := INTI;                      (* RETURN NUMBER *)
END;  (* RDRNUM *)


PROCEDURE BOACMD;                      (* COMMAND - SET UP POSITION *)

VAR
  INTM : TM;                           (* COLOR *)
  INTS : TS;                           (* POSITION ON BOARD *)
```

Listing 1, continued:

```
  PROCEDURE BOAADV(A:TI);                   (* ADVANCE N FILES *)

  BEGIN
    IF INTS+A < ZS THEN
      INTS := INTS+A
    ELSE
      INTS := ZS;
  END;  (* BOAADV *)


  PROCEDURE BOASTO(A:TP);                   (* STORE PIECE ON BOARD *)

  BEGIN
    BOARD.RBIS[INTS] := A;
    IF INTS < ZS THEN
      INTS := INTS+1:
  END:  (* BOASTO *)


BEGIN  (* BOACMD *)
  CLSTAT;                                   (* CLEAR STATUS FLAGS *)
  LSTMV := NULMV:                           (* CLEAR PREVIOUS MOVE *)
  FOR INTS := AS TO ZS DO
    BOARD.RBIS[INTS] := MT;                 (* CLEAR BOARD *)
  INTM := LITE;
  INTS := 0;
  REPEAT
    IF ILINE[JNTJ] IN ["P","R","N","B","Q","K","L","D","1".."8"] THEN
    CASE ILINE[JNTJ] OF
      "P": BOASTO(XTUMP[EP,INTM]);
      "R": BOASTO(XTUMP[ER,INTM]);
      "N": BOASTO(XTUMP[EN,INTM]);
      "B": BOASTO(XTUMP[EB,INTM]);
      "Q": BOASTO(XTUMP[EQ,INTM]);
      "K": BOASTO(XTUMP[EK,INTM]);
      "L": INTM := LITE;
      "D": INTM := DARK;
      "1","2","3","4","5","6","7","8":
           BOAADV(ORD(ILINE[JNTJ])-ORD("0"));
    END
    ELSE
      IF ILINE[JNTJ] IN ["A".."9"] THEN
      BEGIN
        FOR INTS := AS TO ZS DO
          BOARD.RBIS[INTS] := MT;
        CLSTAT;                             (* CLEAR STATUS *)
        RDRERR(" ILLEGAL BOARD OPTION          ");
      END;
    JNTJ := JNTJ+1;
  UNTIL JNTJ = ZJ;
END;  (* BOACMD *)


PROCEDURE ENDCMD;                           (* COMMAND - END PROGRAM *)

BEGIN
  GOTO 9;                                   (* END PROGRAM *)
END;  (* ENDCMD *)


PROCEDURE GONCMD;                           (* COMMAND - GO N MOVES *)

BEGIN
  GOING := RDRNUM;                          (* CRACK NUMBER *)
  IF GOING <= 0 THEN
    GOING := 1;
  GOTO 2;                                   (* EXECUTE MACHINES MOVE *)
END;  (* GONCMD *)


PROCEDURE INICMD;                           (* COMMAND - INITIALIZE FOR A NEW
                                               GAME *)

BEGIN
  GOTO 1;                                   (* INITIALIZE FOR A NEW GAME *)
END;  (* INICMD *)


PROCEDURE LETCMD;                           (* COMMAND - CHANGE VARIABLE *)

LABEL
  21;                                       (* LET COMMAND EXIT *)

PROCEDURE LETONE                            (* TEST FOR AND SET ONE
                                               VARIABLE *)
  (A:RA;                                    (* VARIABLE NAME *)
   VAR B:TI);                               (* VARIABLE *)

BEGIN
  IF A = INRA THEN
  BEGIN
    B := RDRNUM;                            (* GET VALUE *)
    GOTO 21;                                (* EXIT *)
  END;
END;  (* LETONE *)


BEGIN
  IF RDRGNT(INRA) THEN
  BEGIN
    LETONE("FKPSHD    ",FKPSHD);
    LETONE("FKSANQ    ",FKSANQ);
    LETONE("FMAXMT    ",FMAXMT);
    LETONE("FNODEL    ",FNODEL);
    LETONE("FPADQR    ",FPADCR[F1]);
    LETONE("FPADQN    ",FPADCR[F2]);
    LETONE("FPADQB    ",FPADCR[F3]);
    LETONE("FPADQF    ",FPADCR[F4]);
```

```
    LETONE("FPADKF    ",FPADCR[F5]);
    LETONE("FPADKB    ",FPADCR[F6]);
    LETONE("FPADKN    ",FPADCR[F7]);
    LETONE("FPADKR    ",FPADCR[F8]);
    LETONE("FPBLOK    ",FPBLOK);
    LETONE("FPCONN    ",FPCONN);
    LETONE("FPFLNX    ",FPFLNX);
    LETONE("FRDUBL    ",FRDUBL);
    LETONE("FRK7TH    ",FRK7TH);
    LETONE("FTRADE    ",FTRADE);
    LETONE("FTRDSL    ",FTRDSL);
    LETONE("FTRPOK    ",FTRPOK);
    LETONE("FTRPWN    ",FTRPWN);
    LETONE("FWKING    ",FWKING);
    LETONE("FWMAJM    ",FWMAJM);
    LETONE("FWMINM    ",FWMINM);
    LETONE("FWPAWN    ",FWPAWN);
    LETONE("FWROOK    ",FWROOK);
    LETONE("WINDOW    ",WINDOW);
    RDRERR(" ILLEGAL LET VARIABLE NAME     ");
  END;
21:  (* LET COMMAND EXIT *)
END;  (* LETCMD *)


PROCEDURE PLECMD;                           (* COMMAND - PRINT VARIABLE *)

LABEL
  21;                                       (* PRINT LET COMMAND EXIT *)

PROCEDURE PRIONE                            (* TEST FOR AND PRINT VARIABLE *)
  (A:RA;                                    (* TEST VARIABLE NAME *)
   B:TI);                                   (* VARIABLE *)

BEGIN
  IF INRA = A THEN
  BEGIN
    WRITELN(A,B);
    GOTO 21;                                (* EXIT *)
  END;
END;  (* PRIONE *)


BEGIN  (* PLECMD *)
  WHILE RDRGNT(INRA) DO
  BEGIN
    PRIONE("FKPSHD    ",FKPSHD);
    PRIONE("FKSANQ    ",FKSANQ);
    PRIONE("FMAXMT    ",FMAXMT);
    PRIONE("FNODEL    ",FNODEL);
    PRIONE("FPADQR    ",FPADCR[F1]);
    PRIONE("FPADQN    ",FPADCR[F2]);
    PRIONE("FPADQB    ",FPADCR[F3]);
    PRIONE("FPADQF    ",FPADCR[F4]);
    PRIONE("FPADKF    ",FPADCR[F5]);
    PRIONE("FPADKB    ",FPADCR[F6]);
    PRIONE("FPADKN    ",FPADCR[F7]);
    PRIONE("FPADKR    ",FPADCR[F8]);
    PRIONE("FPBLOK    ",FPBLOK);
    PRIONE("FPCONN    ",FPCONN);
    PRIONE("FPFLNX    ",FPFLNX);
    PRIONE("FRDUBL    ",FRDUBL);
    PRIONE("FRK7TH    ",FRK7TH);
    PRIONE("FTRADE    ",FTRADE);
    PRIONE("FTRDSL    ",FTRDSL);
    PRIONE("FTRPOK    ",FTRPOK);
    PRIONE("FTRPWN    ",FTRPWN);
    PRIONE("FWKING    ",FWKING);
    PRIONE("FWMAJM    ",FWMAJM);
    PRIONE("FWMINM    ",FWMINM);
    PRIONE("FWPAWN    ",FWPAWN);
    PRIONE("FWROOK    ",FWROOK);
    PRIONE("WINDOW    ",WINDOW);
    RDRERR(" ILLEGAL VARIABLE NAME         ");

21:  (* PRINT LET CCMMAND EXIT *)
  END;
END;  (* PLECMD *)


PROCEDURE PRICMD;                           (* COMMAND - PRINT BOARD *)

BEGIN
  IF RDRGNT(INRA) THEN
    PRINTB(NBORD)
  ELSE
    PRINTB(BOARD.RBIS);
END;  (* PRICMD *)


PROCEDURE PAMCMD;                           (* COMMAND - PRINT ATTACK MAP *)

BEGIN
  WHILE RDRGNT(INRA) DO
    IF INRA[AA] = "T" THEN
      PRINAM(ATKTO)
    ELSE
      IF INRA[AA] = "F" THEN
        PRINAM(ATKFR)
      ELSE
        RDRERR(" ATTACK MAP NOT 'TO' OR 'FROM'");
END;  (* PAMCMD *)


PROCEDURE POPCMD;                           (* COMMAND - PRINT OTHER STUFF *)

VAR
  INTQ : TQ;                                (* CASTLE TYPE INDEX *)

BEGIN
  WITH BOARD DO
  BEGIN
    WRITELN(XTMA[RBTM]," TO MOVE.");
```

Listing 1, continued:

```
    WRITELN(RBTS," ENPASSANT.");
    WRITELN("MOVE NUMBER",RBTI);
    FOR INTQ := LS TO DL DO
      IF INTQ IN RBSQ THEN
        WRITELN(XTQA[INTQ]," SIDE CASTLE LEGAL.");
  END;
END;  (* POPCMD *)


PROCEDURE PMVCMD;                      (* COMMAND - PRINT MOVE LIST *)

VAR
  INTW : TW;                           (* MOVES LIST INDEX *)

BEGIN
  LSTMOV;                              (* LIST LEGAL MOVES *)
  FOR INTW := AW TO JNTW-1 DO
  BEGIN
    WRITE(INTW:4," ");
    PRIMOV(MOVES[INTW]);
    IF INTW/LPP = INTW DIV LPP THEN
      PAUSER;
  END;
END;  (* PMVCMD *)


PROCEDURE SWICMD;                      (* COMMAND - FLIP SWITCH *)

LABEL
  21;                                  (* SWITCH OPTION EXIT *)


  PROCEDURE SWIONE                     (* PROCESS ONE SWITCH *)
    (A:RA;                             (* SWITCH NAME *)
     VAR B:TB);                        (* SWITCH *)

  VAR
    IMTJ : TJ;                         (* SAVE COMMAND INDEX *)

  BEGIN
    IF INRA = A THEN
    BEGIN
      IMTJ := JNTJ;                    (* SAVE CURRENT POSITION *)
      IF RDRGNT(INRA) THEN
      BEGIN
        IF INRA = "ON        " THEN
          B := TRUE                    (* TURN SWITCH ON *)
        ELSE
          IF INRA = "OFF       " THEN
            B := FALSE                 (* TURN SWITCH OFF *)
          ELSE
            JNTJ := IMTJ;              (* RESTORE CURRENT POSITION *)
        PRISWI(A,B);                   (* PRINT SWITCH VALUE *)
      END
      ELSE
        PRISWI(A,B);
      GOTO 21;                         (* SWITCH OPTION EXIT *)
    END;
  END;  (* SWIONE *)


BEGIN  (* SWICMD *)
21:  (* SWITCH OPTION EXIT *)
  WHILE RDRGNT(INFA) DO
  BEGIN
    SWIONE("EC        ",SWEC);
    SWIONE("PA        ",SWPA);
    SWIONE("PS        ",SWPS);
    SWIONE("RE        ",SWRE);
    SWIONE("SU        ",SWSU);
    SWIONE("TR        ",SWTR);
    RDRERR(" INVALID SWITCH OPTION          ");
  END;
END;  (* SWICMD *)


PROCEDURE STACMD;                      (* COMMAND - STATUS CHANGES *)

LABEL
  21;                                  (* STATUS COMMAND OPTION EXIT *)
VAR
  INRA : RA;                           (* CURRENT TOKEN *)
  INTM : TM;                           (* SIDE BEING PROCESSED *)


  PROCEDURE STAEPF                     (* PROCESS EP FILE *)
    (A:RA;                             (* TEST TOKEN *)
     B:TF);                            (* EQUIVALENT FILE *)

  BEGIN
    IF A = INRA THEN
    BEGIN
      IF INTM = LITE THEN
        BOARD.RBTS := XTRFS[R6,B]
      ELSE
        BOARD.RBTS := XTRFS[R3,B];
      GOTO 21;                         (* EXIT STATUS OPTION *)
    END;
  END;  (* STAEPF *)


  PROCEDURE STACAK;                    (* ALLOW CASTLE KING SIDE *)

  BEGIN
    IF INTM = LITE THEN
      BOARD.RBSQ := BOARD.RBSQ + [LS]
    ELSE
      BOARD.RBSQ := BOARD.RBSQ + [DS];
  END;  (* STACAK *)


  PROCEDURE STACAQ;                    (* ALLOW CASTLE QUEEN SIDE *)

  BEGIN
    IF INTM = LITE THEN
      BOARD.RBSQ := BOARD.RBSQ + [LL]
    ELSE
      BOARD.RBSQ := BOARD.RBSQ + [DL];
  END;  (* STACAQ *)


  PROCEDURE STADRK;                    (* SET BLACK OPTIONS *)

  BEGIN
    INTM := DARK;
  END;  (* STADRK *)


  PROCEDURE STAENP;                    (* SET ENPASSANT FILE *)

  BEGIN
    IF NOT RDRGNT(INRA) THEN
    BEGIN
      CLSTAT;                          (* CLEAR STATUS *)
      RDRERR(" ENPASSANT FILE OMITTED         ");
    END;

    STAEPF("QR        ",F1);
    STAEPF("QN        ",F2);
    STAEPF("QB        ",F3);
    STAEPF("Q         ",F4);
    STAEPF("K         ",F5);
    STAEPF("KB        ",F6);
    STAEPF("KN        ",F7);
    STAEPF("KR        ",F8);
    CLSTAT;                            (* CLEAR STATUS *)
    RDRERR(" ILLEGAL ENPASSANT FILE         ");
  END;  (* STAENP *)


  PROCEDURE STAGOS;                    (* SET SIDE TO MOVE *)

  BEGIN
    BOARD.RBTM := INTM;
    JNTM := INTM;
  END;  (* STAGOS *)


  PROCEDURE STALIT;                    (* SET WHITE OPTIONS *)

  BEGIN
    INTM := LITE;
  END;  (* STALIT *)


  PROCEDURE STANUM;                    (* SET MOVE NUMBER *)

  BEGIN
    BOARD.RBTI := RDRNUM;
  END;  (* STANUM *)


  PROCEDURE STAOPT                     (* TEST STATUS OPTION *)
    (A:RA;                             (* TEST OPTION *)
     PROCEDURE STAXXX);                (* PROCEDURE TO EXECUTE IF
                                            EQUAL *)

  BEGIN
    IF INRA = A THEN
    BEGIN
      STAXXX;                          (* EXECUTE PROCEDURE *)
      GOTO 21;                         (* EXIT STATUS OPTION *)
    END;
  END;  (* STAOPT *)


BEGIN  (* STACMD *)
  CLSTAT;                              (* CLEAR STATUS *)
  INTM := LITE;                        (* DEFAULT SIDE WHITE *)
21:  (* STATUS OPTION EXIT *)
  WHILE RDRGNT(INRA) DO
  BEGIN
    STAOPT("D         ",STADRK);
    STAOPT("EP        ",STAENP);
    STAOPT("G         ",STAGOS);
    STAOPT("L         ",STALIT);
    STAOPT("N         ",STANUM);
    STAOPT("OO        ",STACAK);
    STAOPT("OOO       ",STACAQ);
    CLSTAT;
    RDRERR(" INVALID STATUS OPTION          ");
  END;
END;  (* STACMD *)


PROCEDURE WHACMD;                      (* COMMAND - WHAT? *)

BEGIN
  WRITELN(MOVMS);                      (* PRINT LAST MESSAGE *)
END;  (* WHACMD *)


BEGIN  (* READER *)
11: (* COMMAND EXIT *)
  WHILE NOT RDRMOV DO
    RDLINE;
```

Listing 1, continued:

```
  IF SWEC THEN
  BEGIN                                        (* ECHO LINE *)
    WRITE(" ");
    FOR INTJ := AJ TO ZJ-1 DO
      WRITE(ILINE[INTJ]);
    WRITELN;
  END;
  IF ILINE[AJ+1] IN ["A".."W","Y","Z"] THEN
  BEGIN
    INRA := "        ";                        (* EXTRACT KEYWORD *)
    INRA[AA] := ILINE[AJ];
    INRA[AA+1] := ILINE[AJ+1];
    RDRSFT;                                    (* SKIP FIRST TOKEN *)
    RDRCMD("BO        ",BOACMD);
    RDRCMD("EN        ",ENDCMD);
    RDRCMD("GO        ",GONCMD);
    RDRCMD("IN        ",INICMD);
    RDRCMD("LE        ",LETCMD);
    RDRCMD("PB        ",PAMCMD);
    RDRCMD("PO        ",POPCMD);
    RDRCMD("PL        ",PLECMD);
    RDRCMD("PM        ",PMVCMD);
    RDRCMD("PR        ",PRICMD);
    RDRCMD("ST        ",STACMD);
    RDRCMD("SW        ",SWICMD);
    RDRCMD("WH        ",WHACMD);
    RDRERR(" INVALID COMMAND                  ");
  END;
END;  (* READER *)


PROCEDURE MINENG                               (* GENERATE MINIMUM
                                                  ENGLISH NOTATION *)
  (A:RM;                                       (* MOVE TO NOTATE *)
   B:RA);                                      (* LEADING COMMENT *)

VAR
  INTN : TN;                                   (* MESSAGE INDEX *)

  PROCEDURE ADDCHR                             (* ADD CHARACTER TO MESSAGE *)
    (A:TC);                                    (* CHARACTER *)

  BEGIN
    MOVMS[INTN] := A;                          (* ADD CHARACTER *)
    IF INTN < ZN THEN
      INTN := INTN+1;                          (* ADVANCE POINTER *)
  END;  (* ADDCHR *)


  PROCEDURE ADDSQR                             (* ADD SQUARE TO MESSAGE *)
    (A:TS;                                     (* SQUARE TO ADD *)
     B:RD);                                    (* SQUARE SYNTAX *)

  BEGIN
    WITH B DO
    BEGIN
      IF RDPC THEN
        ADDCHR(XTUC[XTPU[NBORD[A]]]);
      IF RDSL THEN
        ADDCHR("/");
      IF RDKQ THEN
        IF XTSF[A] IN [F1..F4] THEN
          ADDCHR("Q")
        ELSE
          ADDCHR("K");
      IF RDNB THEN
        CASE XTSF[A] OF
          F1,F8: ADDCHR("R");
          F2,F7: ADDCHR("N");
          F3,F6: ADDCHR("B");
          F4   : ADDCHR("Q");
          F5   : ADDCHR("K");
        END;
      IF RDRK THEN
        IF JNTM = LITE THEN
          CASE XTSR[A] OF
            R1: ADDCHR("1");
            R2: ADDCHR("2");
            R3: ADDCHR("3");
            R4: ADDCHR("4");
            R5: ADDCHR("5");
            R6: ADDCHR("6");
            R7: ADDCHR("7");
            R8: ADDCHR("8");
          END
        ELSE
          CASE XTSR[A] OF
            R1: ADDCHR("8");
            R2: ADDCHR("7");
            R3: ADDCHR("6");
            R4: ADDCHR("5");
            R5: ADDCHR("4");
            R6: ADDCHR("3");
            R7: ADDCHR("2");
            R8: ADDCHR("1");
          END;
    END;
  END;  (* ADDSQR *)


  PROCEDURE ADDWRD                             (* ADD WORD TO MESSAGE *)
    (A:RA;                                     (* TEXT OF WORD *)
     B:TA);                                    (* LENGTH OF WORD *)

  VAR
    INTA : TA;                                 (* CHARACTER INDEX *)

  BEGIN
    FOR INTA := AA TO B DO
      ADDCHR(A[INTA]);
  END;  (* ADDWRD *)
```

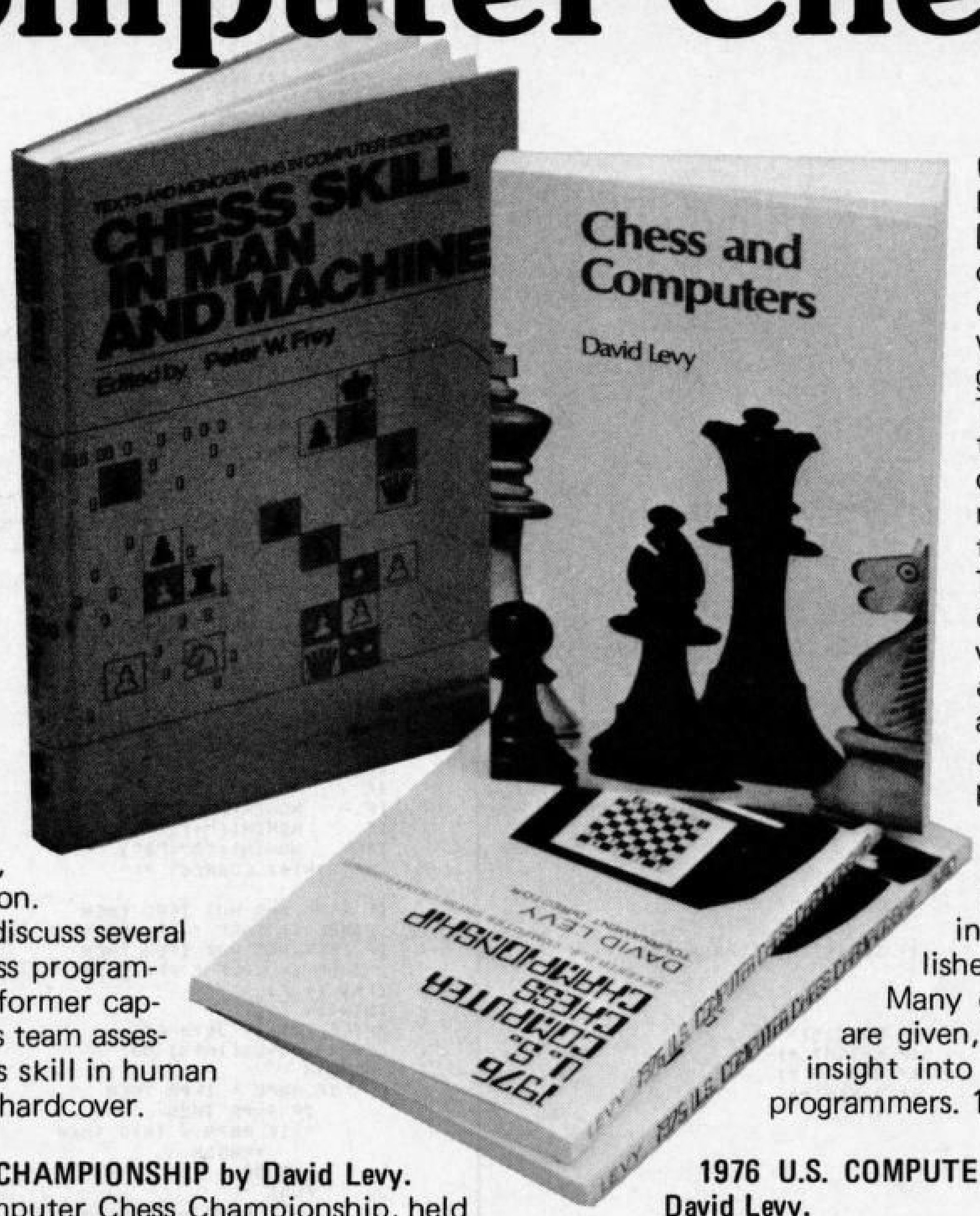
```
  FUNCTION DIFFER                              (* COMPARE MOVES *)
    (A,B:RM)                                   (* MOVES TO COMPARE *)
     :TB;                                      (* TRUE IF MOVES ARE DIFFERENT *)

  VAR
    INTB : TB;                                 (* SCRATCH *)

  BEGIN
    INTB := (A.RMFR <> B.RMFR) OR
            (A.RMTO <> B.RMTO) OR
            (A.RMCP <> B.RMCP);
    IF A.RMPR = B.RMPR THEN
      IF A.RMPR THEN
        DIFFER := INTB OR (A.RMPP <> B.RMPP)
      ELSE
        IF A.RMOQ = B.RMOO THEN
          IF A.RMOO THEN
            DIFFER := INTB OR (A.RMQS <> B.RMQS)
          ELSE
            DIFFER := INTB
        ELSE
          DIFFER := TRUE
    ELSE
      DIFFER := TRUE;
  END;  (* DIFFER *)


  PROCEDURE SETSQD                             (* DEFINE SPECIFIC SQUARE
                                                  DESCRIPTOR *)
    (A:TS;                                     (* SQUARE TO DESCRIBE *)
     B:RD;                                     (* SYNTAX TO USE *)
     VAR C:SR;                                 (* SET OF POSSIBLE RANKS *)
     VAR D:SF);                                (* SET OF POSSIBLE FILES *)

  BEGIN
    C := [R1..R8];                             (* INITIALIZE TO DEFAULTS *)
    D := [F1..F8];
    WITH B DO
    BEGIN
      IF RDKQ AND RDNB THEN
        D := [XTSF[A]];
      IF (NOT RDKQ) AND RDNB THEN
        CASE XTSF[A] OF
          F1,F8: D := [F1,F8];
          F2,F7: D := [F2,F7];
          F3,F6: D := [F3,F6];
          F4   : D := [F4];
          F5   : D := [F5];
        END;
      IF RDRK THEN
        C := [XTSR[A]];
    END;
  END;  (* SETSQD *)


  PROCEDURE MINGEN                             (* PRODUCE MINIMUM
                                                  ENGLISH NOTATION FOR
                                                  MOVES AND CAPTURES *)
    (A:RM;                                     (* MOVE OR CAPTURE *)
     B:TI;                                     (* FIRST SYNTAX TABLE ENTRY *)
     C:TI);                                    (* LAST SYNTAX TABLE ENTRY *)

  LABEL
    21,                                        (* EXIT AMBIGUOUS MOVE SCAN *)
    22;                                        (* EXIT MINGEN *)

  VAR
    INTG : TG;                                 (* PROMOTION PIECE *)
    INTI : TI;                                 (* SYNTAX TABLE INDEX *)
    INTW : TW;                                 (* MOVES INDEX *)
    INLR : SR;                                 (* RANKS DEFINED ON LEFT *)
    INRR : SR;                                 (* RANKS DEFINED ON RIGHT *)
    INLF : SF;                                 (* FILES DEFINED ON LEFT *)
    INRF : SF;                                 (* FILES DEFINED ON RIGHT *)

  BEGIN
    FOR INTI := B TO C DO                      (* FOR EACH SYNTAX ENTRY *)
      WITH SYNTX[INTI] DO
      BEGIN
        IF A.RMPR THEN
          INTG := A.RMPP
        ELSE
          INTG := PB;
        SETSQD(A.RMFR,RYLS,INLR,INLF); (* SET SQUARE SETS *)
        SETSQD(A.RMTO,RYRS,INRR,INRF);
        FOR INTW := AW+1 TO JNTW-1 DO
          IF DIFFER(MOVES[INTW],A) THEN
            IF (NBORD[A.RMFR] = NBORD[MOVES[INTW].RMFR]) AND
               (A.RMCP = MOVES[INTW].RMCP) THEN
              WITH MOVES[INTW] DO
                IF (XTSR[RMFR] IN INLR) AND
                   (XTSR[RMTO] IN INRR) AND
                   (XTSF[RMFR] IN INLF) AND
                   (XTSF[RMTO] IN INRF) AND
                  ((RMPR AND (INTG = RMPP)) OR (NOT RMPR)) THEN
                  GOTO 21;             (* ANOTHER MOVE LOOKS THE SAME *)
        (* NO OTHER MOVE LOOKS THE SAME *)
        ADDSQR(A.RMFR,RYLS);                   (* ADD FROM SQUARE *)
        ADDCHR(RYCH);                          (* ADD MOVE OR CAPTURE *)
        ADDSQR(A.RMTO,RYRS);                   (* ADD TO SQUARE *)
        GOTO 22;                               (* EXIT MINGEN *)
  21:  (* TRY NEXT SYNTAX *)
      END;
  22:  (* EXIT MINGEN *)
  END;  (* MINGEN *)


BEGIN  (* MINENG *)
  MOVMS := "                                     ";
                                               (* CLEAR MESSAGE *)
  INTN := AN+1;                                (* INITIALIZE MESSAGE INDEX *)
  ADDWRD(B,ZA);                                (* ADD INITIAL COMMENT *)
  ADDWRD("-         ",2);
  WITH A DO
  BEGIN
```

Listing 1, continued:

```
    IF RMOO THEN                              (* CASTLE *)
    BEGIN
      ADDWRD("O-O        ",3);
      IF RMQS THEN
        ADDWRD("-O         ",2);
    END
    ELSE                                      (* NOT CASTLE *)
      IF RMCA THEN                            (* CAPTURE *)
        MINGEN(A,SYNCF,SYNCL)
      ELSE                                    (* SIMPLE MOVE *)
        MINGEN(A,SYNMF,SYNML);
    IF RMPR THEN                              (* PROMOTION *)
    BEGIN
      ADDCHR("=");
      ADDCHR(XTGC[RMPP]);
    END;
    ADDWRD(".          ",3);
    IF RMCH THEN                              (* CHECK *)
    BEGIN
      ADDWRD("CHECK      ",5);
      IF RMMT THEN                            (* CHECKMATE *)
        ADDWRD("MATE       ",4);
      ADDCHR(".");
    END
    ELSE
      IF RMMT THEN                            (* STALEMATE *)
        ADDWRD("STALEMATE.",10);
  END;
END;  (* MINENG *)


PROCEDURE MYMOVE;                             (* MAKE MACHINES MOVE *)

VAR
  INRM : RM;                                  (* THE MOVE *)

BEGIN
  CREATE;                                     (* INITIALIZE DATA BASE *)
  INRM := MOVES[SEARCH];                      (* FIND THE BEST MOVE *)
  IF INRM.RMIL THEN
  BEGIN                                       (* NO MOVE FOUND *)
    GOING := 0;
    IF LSTMV.RMCH THEN                        (* CHECKMATE *)
      WRITELN(" CONGRATULATIONS.")
    ELSE                                      (* STALEMATE *)
      WRITELN(" DRAWN. ")
  END
  ELSE
  BEGIN
    MINENG(INRM,"  MY MOVE ");                (* TRANSLATE MOVE TO ENGLISH *)
    WRITELN(MOVMS);                           (* TELL THE PLAYER *)
    THEMOV(INRM);                             (* MAKE THE MOVE *)
    IF SWSU THEN
      WRITELN(BOARD.RBTI,".",NODES," NODES.",BSTVL[AK]);
  END;
END;  (* MYMOVE *)


PROCEDURE YRMOVE;                             (* MAKE PLAYERS MOVE *)

LABEL
  11, 12, 13, 14, 15,                         (* SYNTAX NODES *)
  16,                                         (* SYNTAX ERROR *)
  17,                                         (* AMBIGUOUS MOVE *)
  18;                                         (* NORMAL EXIT *)

VAR
  INTB : TB;                                  (* VALID MOVE FOUND *)
  INTC : TC;                                  (* CURRENT CHARACTER *)
  INTW : TJ;                                  (* MOVES INDEX *)

  INTP : TP;                                  (* MOVING PIECE *)
  INCP : TP;                                  (* CAPTURED PIECE *)
  IFCA : TB;                                  (* CAPTURE *)
  IFPR : TB;                                  (* PROMOTION *)
  IFOO : TB;                                  (* CASTLE *)
  IFQS : TB;                                  (* QUEEN SIDE CASTLE *)
  INTG : TG;                                  (* PROMOTION TYPE *)
  IFMV : TB;                                  (* MOVE FOUND *)

  IFLD : TB;                                  (* R, N, OR B ON LEFT *)
  IFLF : TB;                                  (* K OR Q ON LEFT *)
  IFRD : TB;                                  (* R, N, OR B ON RIGHT *)
  IFRF : TB;                                  (* K OR Q ON RIGHT *)

  INLF : SF;                                  (* FILES ON LEFT *)
  INLR : SR;                                  (* RANKS ON LEFT *)
  INRF : SF;                                  (* FILES ON RIGHT *)
  INRR : SR;                                  (* RANKS ON RIGHT *)

  INRM : RM;                                  (* THE MOVE *)

  FUNCTION NCHIN                              (* DETERMINE IF NEXT INPUT
                                                 CHARACTER IS NOT IN A GIVEN
                                                 SET *)
    (A:SC;                                    (* SET OF CHARACTERS TO CHECK *)
     PROCEDURE YRMXXX)                        (* SEMANTICS ROUTINE TO CALL
                                                 IF NEXT CHARACTER IS IN SET *)
    :TB;                                      (* TRUE IF CHARACTER IS NOT IN
                                                 SET *)

  VAR
    INTB : TB;                                (* SCRATCH *)

  BEGIN
    INTB := NOT (INTC IN A);
    IF NOT INTB THEN
    BEGIN
      YRMXXX;                                 (* EXECUTE SEMANTICS ROUTINE *)
      JNTJ := JNTJ+1;                         (* ADVANCE PAST CHARACTER *)
      WHILE (JNTJ < ZJ)
        AND ((ILINE[JNTJ] = " ") OR (ORD(ILINE[JNTJ]) > ORD(ZC))) DO
```

```
        JNTJ := JNTJ+1;                     (* SKIP BLANKS *)
      INTC := ILINE[JNTJ];                  (* NEXT CHARACTER *)
      IF (INTC = ".") OR (INTC = ";") THEN
        GOTO 15;                            (* EXIT SCAN *)
    END;
    NCHIN := INTB;                          (* RETURN TRUE IF CHARACTER IS
                                               NOT IN STRING *)
  END;  (* NCHIN *)


  PROCEDURE YRMHIT;                         (* FOUND A MOVE.  EXITS
                                               TO AMBIGUOUS MOVE IF THIS
                                               IS THE SECOND POSSIBLE MOVE.
                                               SAVES THE MOVE IN INRM
                                               OTHERWISE.  *)

  BEGIN
    IF IFMV THEN GOTO 17;                   (* SECOND POSSIBLE MOVE *)
    IFMV := TRUE;                           (* FIRT POSSIBLE MOVE *)
    INRM := MOVES[INTW];                    (* SAVE MOVE *)
  END;  (* YRMHIT *)


  PROCEDURE YRMCOM;                         (* COMPARE SQUARES.  CALLS YRMHIT
                                               IF MOVES[INTW] MOVES THE
                                               RIGHT TYPE OF PIECE, CAPTURES
                                               THE RIGHT TYPE OF PIECE, AND
                                               MOVES TO AND FROM POSSIBLE
                                               SQUARES *)

  BEGIN
    WITH MOVES[INTW] DO
      IF (XTSR[RMFR] IN INLR) AND
         (XTSF[RMFR] IN INLF) AND
         (XTSR[RMTO] IN INRR) AND
         (XTSF[RMTO] IN INRF) AND
         (NOT RMIL) AND
         (BOARD.RBIS[RMFR] = INTP) THEN
        IF RMCA = IFCA THEN
          IF RMCA THEN
            IF RMCP = INCP THEN
              YRMHIT
            ELSE
          ELSE
            YRMHIT;
  END;  (* YRMCOM *)


  PROCEDURE YRMCAP;                         (* SEMANTICS - CAPTURE *)

  BEGIN
    IFCA := TRUE;
  END;  (* YRMCAP *)


  PROCEDURE YRMCAS;                         (* SEMANTICS - CASTLE *)

  BEGIN
    IFOO := TRUE;
  END;  (* YRMCAS *)


  PROCEDURE YRMCPC;                         (* SEMANTICS - CAPTURED PIECE *)

  BEGIN
    CASE INTC OF
     "P": INCP := XTUMP[EP,OTHER[JNTM]];
     "R": INCP := XTUMP[ER,OTHER[JNTM]];
     "N": INCP := XTUMP[EN,OTHER[JNTM]];
     "B": INCP := XTUMP[EB,OTHER[JNTM]];
     "Q": INCP := XTUMP[EQ,OTHER[JNTM]];
    END;
  END;  (* YRMCPC *)


  PROCEDURE YRMCQS;                         (* SEMANTICS - CASTLE LONG *)

  BEGIN
    IFQS := TRUE;
  END;  (* YRMCQS *)


  PROCEDURE YRMLKQ;                         (* SEMANTICS - K OR Q ON LEFT *)

  BEGIN
    CASE INTC OF
     "K": INLF := [F5..F8] * INLF;          (* KING SIDE *)
     "Q": INLF := [F1..F4] * INLF;          (* QUEEN SIDE *)
    END;
    IFLF := TRUE;
  END;  (* YRMLKQ *)


  PROCEDURE YRMLRB;                         (* SEMANTICS - R, N, OR B ON
                                               LEFT *)

  BEGIN
    CASE INTC OF
     "R":  INLF := [F1,F8] * INLF;          (* ROOK FILE *)
     "N":  INLF := [F2,F7] * INLF;          (* KNIGHT FILE *)
     "B":  INLF := [F3,F6] * INLF;          (* BISHOP FILE *)
    END;
    IFLD := TRUE;
  END;  (* YRMLRB *)


  PROCEDURE YRMLRK;                         (* SEMANTICS - RANK ON LEFT *)

  BEGIN
    IF JNTM = LITE THEN
```

Computer Chess


TEXTS AND MONOGRAPHS IN COMPUTER SCIENCE
CHESS SKILL IN MAN AND MACHINE
Edited by Peter W. Frey
Chess and Computers
David Levy
1976 U.S. COMPUTER CHESS CHAMPIONSHIP
DAVID LEVY
LEVY 1975 U.S. COMPUTER CHESS CHAMPIONSHIP

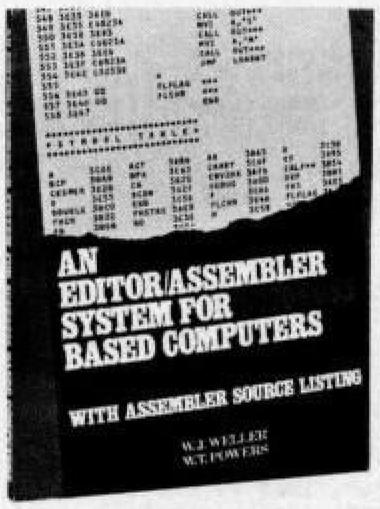
CHESS SKILL IN MAN AND MACHINE edited by Peter W Frey.
□ A game of endless variations, chess has challenged our skill for centuries. This book surveys our current understanding of human chess skill and covers the subtleties of coaxing a machine to play chess. The initial chapter and appendix present a brief history of the computer chess tournaments. The next two chapters describe the essentials of how humans and computers play chess. The fourth chapter provides a detailed description of the Northwestern Chess Program, currently the national champion. The following three chapters discuss several alternative approaches to chess programming. In the final chapter, a former captain of the U.S. Olympic chess team asseses the present status of chess skill in human and machine. 217 pp. $14.80 hardcover.

CHESS AND COMPUTERS by David Levy.
□ If you enjoy playing chess, then you will thoroughly enjoy this book, which is loaded with chess games played by computers. The first chapter describes the earliest chess "machine," the famous Automaton chess player that toured Europe and America. There is a detailed account of Torres y Quevedo's invention that played the ending of King and Rook against King. There is also a description of how computers play chess, including an account of early Soviet attempts at chess programming that contains much information hitherto unpublished outside the Soviet Union. Many examples of computer play are given, which provide an excellent insight into the problems facing chess programmers. 145 pp. $8.95.

1975 U.S. COMPUTER CHESS CHAMPIONSHIP by David Levy.
□ The sixth annual U.S. Computer Chess Championship, held in October 1975, was a tournament in which twelve computer programs competed against each other. This book includes a detailed analysis and description of all the games, presented by David Levy, the tournament director. 86 pp. $5.95.

1976 U.S. COMPUTER CHESS CHAMPIONSHIP by David Levy.
□ This book includes a detailed analysis and description of all the tournament games played at the seventh annual U.S. Chess Championship held in October 1976. 90 pp. $5.95.

A New Book!


AN EDITOR/ASSEMBLER SYSTEM FOR BASED COMPUTERS
WITH ASSEMBLER SOURCE LISTING
W.J. WELLER
W.T. POWERS

AN EDITOR/ASSEMBLER SYSTEM FOR 8080/8085 BASED COMPUTERS by W J Weller and W T Powers.
□ This 148-page book contains complete information for initializing and using a powerful new editor/assembler and debugging monitor system, and the full SOURCE text of both. The assembler fully supports all Intel instruction mnemonics as well as the entire language used in PRACTICAL MICROCOMPUTER PROGRAMMING™: THE INTEL 8080. The editor/assembler is resident in less than 8K RAM and will run on any 8080, 8085 or Z80 based computer with peripherals which transfer on a character by character basis or can be made to do so by buffering. The user supplies his own I/O drivers. The text editor is extremely simple to use and does not require irrelevant line numbers. Also included is a program to convert Processor Technology™ format tapes to a format useable by the editor/assembler.

This system is not the usual "quickie" software, riddled with errors and limitations, but a professionally created, thoroughly tested and debugged system. At $14.95 it is the best software bargain you are ever likely to see.

AND BY THE WAY. . . paper tape object copies of all of this software are sent FREE to book purchasers when the coupon at the back of the book is returned to Northern Technology Books. 8½ x 11 format. $14.95.

For your convenience in ordering, please use this page plus the order form on page 138.

DIAL YOUR BANK CARD ORDERS TOLL FREE (800) 258-5477.

BITS™ inc

STANDARD DICTIONARY OF COMPUTERS AND INFORMATION PROCESSING 2nd Edition by Martin H Weik.
□ This is a very complete, fully cross-referenced dictionary. It goes a step farther in that it includes full explanations, practical examples, many pertinent illustrations, and supplementary information for over 12,500 hardware and software terms. It cross-references the terms to other closely related concepts, and appended to each definition, as the need arises, are explanations, tutorial information, examples, usage areas, and cross-references for further clarification of concepts and meanings. 390 pp. $16.95 hardcover. (No photo)

Circle 35 on inquiry card.

Listing 1, continued:

```
    CASE INTC OF
     "1": INLR := [R1];
     "2": INLR := [R2];
     "3": INLR := [R3];
     "4": INLR := [R4];
     "5": INLR := [R5];
     "6": INLR := [R6];
     "7": INLR := [R7];
     "8": INLR := [R8];
    END
  ELSE
    CASE INTC OF
     "1": INLR := [R8];
     "2": INLR := [R7];
     "3": INLR := [R6];
     "4": INLR := [R5];
     "5": INLR := [R4];
     "6": INLR := [R3];
     "7": INLR := [R2];
     "8": INLR := [R1];
    END;
END;  (* YRMLRK *)


PROCEDURE YRMNUL;                     (* SEMANTICS - NULL *)

BEGIN
END;  (* YRMNUL *)


PROCEDURE YRMPCM;                     (* SEMANTICS - PIECE MOVED *)

BEGIN
  CASE INTC OF
   "P": INTP := XTUMP[EP,JNTM];       (* PAWN *)
   "R": INTP := XTUMP[ER,JNTM];       (* ROOK *)
   "N": INTP := XTUMP[EN,JNTM];       (* KNIGNT *)
   "B": INTP := XTUMP[EB,JNTM];       (* BISHOP *)
   "Q": INTP := XTUMP[EQ,JNTM];       (* QUEEN *)
   "K": INTP := XTUMP[EK,JNTM];       (* KING *)
  END;
END;  (* YRMPCM *)


PROCEDURE YRMPRO;                     (* SEMANTICS - PROMOTION *)

BEGIN
  CASE INTC OF
   "R": INTG := PR;                   (* ROOK *)
   "N": INTG := PN;                   (* KNIGHT *)
   "B": INTG := PB;                   (* BISHOP *)
   "Q": INTG := PQ;                   (* QUEEN *)
  END;
  IFPR := TRUE;
END;  (* YRMPRO *)


PROCEDURE YRMRKQ;                     (* SEMANTICS - K OR Q ON RIGHT *)

BEGIN
  CASE INTC OF
   "K": INRF := [F5..F8] * INRF;      (* KING SIDE *)
   "Q": INRF := [F1..F4] * INRF;      (* QUEEN SIDE *)
  END;
  IFRF := TRUE;
END;  (* YRMLKQ *)


PROCEDURE YRMRRB;                     (* SEMANTICS - R, N, OR B ON
                                         RIGHT *)

BEGIN
  CASE INTC OF
   "R":  INRF := [F1,F8] * INRF;      (* ROOK FILE *)
   "N":  INRF := [F2,F7] * INRF;      (* KNIGHT FILE *)
   "B":  INRF := [F3,F6] * INRF;      (* BISHOP FILE *)
  END;
  IFRD := TRUE;
END;  (* YRMLRB *)


PROCEDURE YRMRRK;                     (* SEMANTICS - RANK ON RIGHT *)

BEGIN
  IF JNTM = LITE THEN
    CASE INTC OF
     "1": INRR := [R1];
     "2": INRR := [R2];
     "3": INRR := [R3];
     "4": INRR := [R4];
     "5": INRR := [R5];
     "6": INRR := [R6];
     "7": INRR := [R7];
     "8": INRR := [R8];
    END
  ELSE
    CASE INTC OF
     "1": INRR := [R8];
     "2": INRR := [R7];
     "3": INRR := [R6];
     "4": INRR := [R5];
     "5": INRR := [R4];
     "6": INRR := [R3];
     "7": INRR := [R2];
     "8": INRR := [R1];
    END;
END;  (* YRMLRK *)
```

```
BEGIN  (* YRMOVE *)
  INTB := FALSE;
  WHILE NOT INTB DO
  BEGIN
    READER;                               (* READ NEXT MOVE *)
    LSTMOV;                               (* LIST LEGAL MOVES *)
    IFCA := FALSE;
    IFPR := FALSE;
    IFOO := FALSE;
    IFQS := FALSE;
    IFLD := FALSE;
    IFLF := FALSE;
    IFRD := FALSE;
    IFRF := FALSE;
    INTP := MT;
    INCP := MT;
    INLF := [F1..F8];
    INRF := [F1..F8];
    INLR := [R1..R8];
    INRR := [R1..R8];

    INTC := ILINE[JNTJ];

    IF     NCHIN(["P","R","N","B","Q","K"],YRMPCM) THEN GOTO 14;
    IF     NCHIN(["/"]                    ,YRMNUL) THEN GOTO 11;
    IF     NCHIN(["K","Q"]                ,YRMLKQ) THEN;
    IF     NCHIN(["R","N","B"]            ,YRMLRB) THEN;
    IF     NCHIN(["1".."8"]               ,YRMLRK) THEN;
11:  (* LEFT SIDE DONE *)
    IF NOT NCHIN(["-"]                    ,YRMNUL) THEN GOTO 12;
    IF     NCHIN(["*","X"]                ,YRMCAP) THEN GOTO 16;
    IF     NCHIN(["P","R","N","B","Q"]    ,YRMCPC) THEN GOTO 16;
    IF     NCHIN(["/"]                    ,YRMNUL) THEN GOTO 13;
12:  (* RIGHT SIDE SQUARE *)
    IF     NCHIN(["K","Q"]                ,YRMRKQ) THEN;
    IF     NCHIN(["R","N","B"]            ,YRMRRB) THEN;
    IF     NCHIN(["1".."8"]               ,YRMRRK) THEN;
13:  (* PROMOTION *)
    IF     NCHIN(["="]                    ,YRMNUL) THEN GOTO 15;
    IF     NCHIN(["R","N","B","Q"]        ,YRMPRO) THEN GOTO 16;
    GOTO 15;

14:  (* CASTLING *)
    IF     NCHIN(["O","0"]                ,YRMNUL) THEN GOTO 16;
    IF     NCHIN(["-"]                    ,YRMNUL) THEN GOTO 16;
    IF     NCHIN(["O","0"]                ,YRMCAS) THEN GOTO 16;
    IF     NCHIN(["-"]                    ,YRMCQS) THEN GOTO 15;
    IF     NCHIN(["O","0"]                ,YRMNUL) THEN GOTO 16;
15:  (* SYNTAX CORRECT *)

    IF IFRF AND NOT IFRD THEN
      INRF := INRF * [F4,F5];             (* SELECT K OR Q FILE *)
    IF IFLF AND NOT IFLD THEN
      INLF := INLF * [F4,F5];             (* SELECT K OR Q FILE *)
    IFMV := FALSE;                        (* NO MOVE FOUND YET *)
    INTW := AW;                           (* INITIALIZE INDEX *)
    WHILE INTW < JNTW DO
      WITH MOVES[INTW] DO
      BEGIN
        IF RMPR = IFPR THEN
          IF RMPR THEN
            IF RMPP = INTG THEN           (* CORRECT PROMOTION TYPE *)
              YRMCOM                      (* COMPARE SQUARES AND PIECES *)
            ELSE
          ELSE                            (* NOT PROMOTION *)
            IF RMOO = IFOO THEN
              IF RMOO THEN                (* CASTLING *)
                IF RMQS = IFQS THEN       (* CASTLING SAME WAY *)
                  YRMHIT
                ELSE
              ELSE                        (* NOT CASTLING *)
                YRMCOM;                   (* COMPARE SQUARES AND PIECES *)
        INTW := INTW+1;                   (* ADVANCE MOVES INDEX *)
      END;
    IF IFMV THEN                          (* ONE MOVE FOUND *)
    BEGIN
      MINENG(INRM,"YOUR MOVE ");          (* CONVERT TO OUR STYLE *)
      WRITELN(MOVMS);                     (* PRINT MOVE *)
      THEMOV(INRM);                       (* MAKE THE MOVE *)
      INTB := TRUE;                       (* EXIT YRMOVE *)
    END
    ELSE                                  (* NO MOVES FOUND *)
      WRITELN(" ILLEGAL MOVE.");
    GOTO 18;                              (* EXIT *)

16:  (* SYNTAX ERROR *)
    WRITELN(" SYNTAX ERROR.");
    GOTO 18;                              (* EXIT *)

17:  (* AMBIGUOUS MOVE *)
    WRITELN(" AMBIGUOUS MOVE.");
18:  (* EXIT *)
  END;
END;  (* YRMOVE *)


BEGIN  (* THE PROGRAM *)
  WRITELN(" HI.  THIS IS CHESS .5");
  INICON;                                 (* INITIALIZE CONSTANTS *)

1:  (* INITIALIZE FOR A NEW GAME *)
  INITAL (BOARD);                         (* INITIALIZE FOR A NEW GAME *)
  REPEAT
    REPEAT
      YRMOVE;                             (* EXECUTE PLAYERS MOVE *)
    UNTIL SWRE;

2:  (* EXECUTE MACHINES MOVE *)
    REPEAT
      MYMOVE;
      IF GOING > 0 THEN
        GOING := GOING-1;
    UNTIL GOING = 0;
  UNTIL FALSE;

9:  (* END OF PROGRAM *)
END.
```

Text continued from page 144

A second and somewhat more challenging project would be to develop a transposition table for the program. This requires the availability of unused memory (at least 8 K bytes and preferably 16 K or 32 K bytes), an efficient hashing scheme, and a set of decision rules to select among positions when a collision occurs (ie: two positions hash to the same address in the table). Another problem is that the use of a staged evaluation process and the α-β algorithm often provides an imprecise evaluation score (ie: the machine has determined that a position was not optimal but has not invested the time to find out exactly how bad it was). If the programmer succeeds with the transposition table, however, move calculation will take 30 to 50 per cent less time in most middle game positions and 60 to 90 per cent less time in many end game positions.

A third area for improvement is the evaluation function. Our program presently has only a rudimentary function. The reader should compare it with the one used by Chess 4.5 which is described in detail by Slate and Atkin. Their evaluation function provides an excellent starting point for revising our present function. In part 4 we will discuss the advantages of using a conditional evaluation function, ie: one that changes depending on the stage of the game and on the presence of special features. One implementation of this strategy is the special end game program described by Monroe Newborn in *Chess Skill in Man and Machine.*

It is appropriate for us to add two important disclaimers at this juncture. Although we have carefully tested each of the routines in the program and played several chess games, it is still possible that there are a few minor bugs in the program. If you find one, a letter to one of us or to BYTE would be appreciated. Secondly, our chess program was written primarily for pedagogical purposes. For this reason it is not a production program and does not run very efficiently. If you are the competitive type, our program should provide many useful ideas, but you should not expect it to compete successfully in tournament play unless you make extensive modifications and additions.

A chess program has a tendency to grow and change its personality as the programmer becomes more familiar with each of its many limitations. It provides a constant challenge for those of us who are too compulsive to tolerate obvious weaknesses. In fact one must be careful not to become totally obsessed with this project. We do not wish any of you to lose your job or your spouse because of a chess program.■