Listing 1 (opposite): The first half of Chess 0.5, written in Pascal. The second half of the program will be presented in part 3 (December 1978 BYTE) of this series. The portion of the program presented here covers initialization of the program, variable declaration, manipulation of the "bit boards" (used to represent positions on the chessboard), user print routines and move generation. The second half of the listing will include procedures for evaluation of terminal positions, the look-ahead procedure, and user commands.

Creating a Chess Player Part 2: Chess 0.5

Part 1 of this series ("Creating a Chess Player," October 1978 BYTE, page 182) was an essay on human and computer skill. This month and next we present Chess 0.5, a program written in Pascal by Larry Atkin, who is coauthor with David Slate of the world championship computer 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 of the series will conclude with some thoughts about computer chess strategy.

Peter W Frey
Dept of Psychology
Northwestern University
Evanston IL 60201

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

Note: *The Pascal subset described in "A 'Tiny' Pascal Compiler" (page 182) is not compatible with the more sophisticated Pascal used here. . . .*CM

We have attempted to incorporate several features which make the search process more efficient and others which increase the user's options. Both of these enhancements are important. The first set of features (incremental updating, iterative searching, staged move generation, etc) were described in general terms in part 1. These features reduce computation to the point where a move can be selected in a reasonable amount of time even with a full-width search. The second set of features (special control and print commands, accepting chess moves in standard notation) not only add to the pleasure of using the program, but also make the debugging process much easier. The price for these enhancements is a longer, more complicated program. We hope the length of our listing will not discourage the reader from becoming actively involved.

Pascal was developed to provide a logical and systematic higher level language which could produce reasonably efficient machine code for existing hardware. Computer programs can be conceptualized in terms of two essential parts, descriptions of data and descriptions of actions which are to be performed on the data. Pascal requires that every variable occurring in the program be introduced by a declaration statement which associates an identifier and a data type with that variable. The data type defines the set of values which may be assumed by the variable. Since a chess program involves a large number of variables, our program begins with a long list of declaration statements.

A constant definition introduces an identifier as a synonym for a constant. This is very useful since the value of the constant as stated in the declaration list can be changed at some later date, and this change will then be reflected throughout the program in every place where the constant is used. In the chess program, the values of some of the constants depend on the characteristics of the user's hardware. For example, the values of ZK (maximum search depth) and ZW (move stack limit) will reflect the amount of memory which is available on your system. On personal computers, ZX will generally be set at 7 if you have an 8 bit processor and at 15 if you have a 16 bit processor. Note also that the value of PZX8 depends on the value of ZX. To implement this program on a given computer, it is necessary to insert at the beginning of the program the appropriate values for these constants.

For the sake of clarity, specific data types are declared for a number of different chess concepts and for certain useful indices. The program also takes advantage of the different properties represented in Pascal's data structures: the set, array and record. It is unlikely that anyone will immediately memorize the names of all the variables. Therefore it is useful to have them listed at the beginning where they can easily be found for later reference.

There is a comment statement accompanying almost every instruction in the program. Although these brief statements

```
PROGRAM CHESS(INPUT,OUTPUT);

LABEL

  1,                                    (* INITIALIZE FOR A NEW GAME *)
  2,                                    (* EXECUTE MACHINES MOVE *)
  9;                                    (* END OF PROGRAM *)



CONST

  AA = 1; ZA = 10;                      (* CHARACTERS IS A WORD *)
  AC = "A"; ZC = "Z";                   (* CHARACTER LIMITS *)
  AD = -21; ZD = +21;                   (* DIRECTION LIMITS *)
  AJ = 0; ZJ = 73;                      (* CHARACTERS IN A STRING *)
  AK = 0; ZK = 16;                      (* SEARCH DEPTH LIMITS *)
  AKM2 = -2;                            (* AK-2 *)
  ZKP1 = 17;                            (* ZK+1 *)
  AL = 0; ZL = 119;                     (* LARGE BOARD VECTOR LIMITS *)
  AZL = -119; ZAL = 119;                (* LARGE BOARD DIFFERENCES
                                           LIMITS *)
  AN = 1; ZN = 30;                      (* MESSAGE LIMITS *)
  AS = 0; ZS = 63;                      (* BOARD VECTOR LIMITS *)
  AT = -1; ZT = 63;                     (* BOARD VECTOR LIMITS AND
                                           ANOTHER VALUE *)
  AV = -32767; ZV = +32767;             (* EVALUATION LIMITS *)
  AW = 1; ZW = 500;                     (* MOVE STACK LIMITS *)
  AX = 0; ZX = 31;                      (* SUBSETS OF SQUARES *)
  AY = 0; ZY = 1;                       (* ARRAY OF SUBSETS TO FORM A SET
                                           OF ALL SQUARES ON BOARD *)

  LPP = 20;                             (* LINES PER PAGE *)
  PZX8 = 16777216;                      (* 2^(ZX-7) *)

  SYNCF = 1;                            (* FIRST CAPTURE SYNTAX *)
  SYNCL = 36;                           (* LAST CAPTURE SYNTAX *)
  SYNMF = 37;                           (* FIRST MOVE SYNTAX *)
  SYNML = 47;                           (* LAST MOVE SYNTAX *)


TYPE

  (* SIMPLE TYPES *)

  TA = AA..ZA;                          (* INDEX TO WORDS OF CHAR *)
  TB = BOOLEAN;                         (* TRUE OR FALSE *)
  TC = CHAR;                            (* SINGLE CHARACTERS *)
  TD = AD..ZD;                          (* DIRECTIONS *)
  TE = (E1,E2,E3,E4,S1,S2,S3,S4,N1,N2,N3,N4,N5,N6,N7,N8);
                                        (* NUMBER OF DIRECTIONS *)
  TF = (F1,F2,F3,F4,F5,F6,F7,F8);       (* FILES *)
  TG = (PQ,PR,PN,PB);                   (* PROMOTION PIECES *)
  TH = (H0,H1,H2,H3,H4,H5,H6,H7);
                                        (* TREE SEARCH MODES *)
  TI = INTEGER;                         (* NUMBERS *)
  TJ = AJ..ZJ;                          (* INDEX TO STRINGS *)
  TK = AK..ZK;                          (* PLY INDEX *)
  TL = AL..ZL;                          (* LARGE (10X12) BOARD *)
  TM = (LITE,DARK,NONE);                (* SIDES *)
  TN = AN..ZN;                          (* INDEX TO MESSAGES *)
  TP = (LP,LR,LN,LB,LQ,LK,DP,DR,DN,DB,DQ,DK,MT);
                                        (* PIECES:  LIGHT PAWN, LIGHT
                                           ROOK, ... , DARK KING, EMPTY
                                           SQUARE *)
  TQ = (LS,LL,DS,DL);                   (* QUADRANTS *)
  TR = (R1,R2,R3,R4,R5,R6,R7,R8);       (* RANKS *)
  TS = AS..ZS;                          (* SQUARES *)
  TT = AT..ZT;                          (* SQUARES, AND ANOTHER VALUE *)
  TU = (EP,ER,EN,EB,EQ,EK);             (* TYPES: PAWN, ROOK, ... ,
                                           KING *)
  TV = AV..ZV;                          (* EVALUATIONS *)
  TW = AW..ZW;                          (* MOVES INDEX *)
  TX = AX..ZX;                          (* SOME SQUARES *)
  TY = AY..ZY;                          (* NUMBER OF TX'S IN A BOARD *)
  TZ = REAL;                            (* FLOATING POINT NUMBERS *)

  (* SETS *)

  SC = SET OF AC..ZC;                   (* SET OF CHARACTERS *)
  SF = SET OF TF;                       (* SET OF FILES *)
  SQ = SET OF TQ;                       (* SET OF CASTLING TYPES *)
  SR = SET OF TR;                       (* SET OF RANKS *)
  SX = SET OF TX;                       (* SET OF SOME SQUARES *)

  (* RECORDS *)

  RB = RECORD                           (* BOARDS *)
    RBTM : TM;                          (* SIDE TO MOVE *)
    RBTS : TT;                          (* ENPASSANT SQUARE *)
    RBTI : TI;                          (* MOVE NUMBER *)
    RBSQ : SQ;                          (* CASTLE FLAGS *)
    CASE INTEGER OF
      0: ( RBIS: ARRAY (TS) OF TP);     (* INDEXED BY SQUARE *)
      1: ( RBIRF: ARRAY (TR,TF) OF TP);(* INDEXED BY RANK AND FILE *)
    END;

  RA = PACKED ARRAY (TA) OF TC;         (* WORDS OF CHARACTERS *)
  RC = ARRAY (TS) OF TP;                (* BOARD VECTORS *)
  RN = PACKED ARRAY (TN) OF TC;         (* MESSAGES *)
  RJ = PACKED ARRAY (TJ) OF TC;         (* STRINGS *)

  RD = PACKED RECORD                    (* SYNTAX DESCRIPTOR FOR
                                           SINGLE SQUARE *)
    RDPC : TB;                          (* PIECE *)
    RDSL : TB;                          (* / *)
    RDKQ : TB;                          (* K OR Q *)
    RDNB : TB;                          (* R, N, OR B *)
    RDRK : TB;                          (* RANK *)
    END;

  RK = RECORD                           (* KLUDGE TO FIND NEXT BIT *)
    CASE INTEGER OF
      0: (RKTB: SET OF 0..47);          (* BITS *)
      1: (RKTZ: TZ);                    (* FLOATING POINT NUMBER *)
    END;

  RM = PACKED RECORD                    (* MOVES *)
    RMFR : TS;                          (* FROM SQUARE *)
    RMTO : TS;                          (* TO SQUARE *)
    RMCP : TP;                          (* CAPTURED PIECE *)
    RMCA : TB;                          (* CAPTURE *)
    RMAC : TB;                          (* AFFECTS CASTLE STATUS *)
    RMCH : TB;                          (* CHECK *)
    RMMT : TB;                          (* MATE *)
    RMIL : TB;                          (* ILLEGAL *)
    RMSU : TB;                          (* SEARCHED *)
    CASE RMPR : TB OF                   (* PROMOTION *)
      FALSE: (
        CASE RMOO : TB OF               (* CASTLE *)
          FALSE: (RMEP : TB);           (* ENPASSANT *)
          TRUE : (RMQS : TB);           (* QUEEN SIDE *)
          );
      TRUE : (RMPP : TG);               (* PROMOTION TYPE *)
    END;

  RS = RECORD                           (* BIT BOARDS *)
    CASE INTEGER OF
      0: (RSSS: ARRAY (TY) OF SX);      (* ARRAY OF SETS *)
      1: (RSTI: ARRAY (TY) OF TI);      (* ARRAY OF INTEGERS *)
    END;

  RX = ARRAY (TS) OF RS;                (* ATTACK MAPS *)

  RY = PACKED RECORD                    (* MOVE SYNTAX DESCRIPTOR *)
    RYLS : RD;                          (* LEFT SIDE DESCRIPTOR *)
    RYCH : TC;                          (* MOVE OR CAPTURE *)
    RYRS : RD;                          (* RIGHT SIDE DESCRIPTOR *)
    END;

  RE = ARRAY (TW) OF TV;                (* ARRAY OF VALUES *)
  RF = ARRAY (TW) OF RM;                (* ARRAY OF MOVES *)

VAR

  (* DATA BASE *)

  BOARD : RB;                           (* THE BOARD *)
  NBORD : ARRAY (TS) OF TP;             (* LOOK-AHEAD BOARD *)
  ATKFR : ARRAY (TS) OF RS;             (* ATTACKS FROM A SQUARE *)
  ATKTO : ARRAY (TS) OF RS;             (* ATTACKS TO A SQUARE *)
  ALATK : ARRAY (TM) OF RS;             (* ATTACKS BY EACH COLOR *)
  TPLOC : ARRAY (TP) OF RS;             (* LOCATIONS OF PIECE BY TYPE *)
  TMLOC : ARRAY (TM) OF RS;             (* LOCATIONS OF PIECE BY COLOR *)
  MOVES : ARRAY (TW) OF RM;             (* MOVES *)
  VALUE : ARRAY (TW) OF TV;             (* VALUES *)
  ALLOC : ARRAY (TK) OF RS;             (* ALL PIECES *)
  BSTMV : ARRAY (TK) OF TW;             (* BEST MOVE SO FAR *)
  BSTVL : ARRAY (AKM2..ZKP1) OF TV;     (* VALUE OF BEST MOVE *)
  CSTAT : ARRAY (TK) OF RS;             (* CASTLING SQUARES *)
  ENPAS : ARRAY (TK) OF RS;             (* ENPASSANT SQUARES *)
  GENPN : ARRAY (TK) OF RS;             (* PAWN ORIGINATION SQUARES *)
  GENTO : ARRAY (TK) OF RS;             (* MOVE DESTINATION SQUARES *)
  GENFR : ARRAY (TK) OF RS;             (* MOVE ORIGINATION SQUARES *)
  MBVAL : ARRAY (TK) OF TV;             (* MATERIAL BALANCE VALUES *)
  MVSEL : ARRAY (TK) OF TI;             (* COUNT MOVES SELECTED BY PLY *)
  INDEX : ARRAY (AK..ZKP1) OF TW;       (* CURRENT MOVE FOR PLY *)
  KILLR : ARRAY (TK) OF RM;             (* KILLER MOVES BY PLY *)
  LINDX : ARRAY (TK) OF TW;             (* LAST MOVE FOR PLY *)
  SRCHM : ARRAY (TK) OF TH;             (* SEARCH MODES *)
  GOING : TI;                           (* NUMBER OF MOVES TO EXECUTE *)
  LSTMV : RM;                           (* PREVIOUS MOVE *)
  MAXPS : TV;                           (* MAXIMUM POSITIONAL SCORE *)
  MBLTE : TV;                           (* MATERIAL BALANCE LITE EDGE *)
  MBPWN : ARRAY (TM) OF TI;             (* NUMBER OF PAWNS BY SIDE *)
  MBTOT : TV;                           (* TOTAL MATERIAL ON MBORD *)
  NODES : TI;                           (* NUMBER OF NODES SEARCHED *)

  JNTK  : TK;                           (* PLY INDEX *)
  JNTK  : TK;                           (* ITERATION *)
  JNTM  : TM;                           (* SIDE TO MOVE *)
  JNTW  : TW;                           (* MOVES STACK POINTER *)

  (* LETS *)

  FKPSHD : TI;                          (* KING PAWN SHIELD CREDIT *)
  FKSANQ : TI;                          (* KING IN SANCTUARY CREDIT *)
  FMAXMT : TI;                          (* MAXIMUM MATERIAL SCORE *)
  FNODEL : TI;                          (* NODE LIMIT FOR SEARCH *)
  FPADCR : ARRAY (TF) OF TI;            (* PAWN ADVANCE CREDIT BY FILE *)
  FPBLOK : TI;                          (* PAWN BLOCKED PENALTY *)
  FPCONN : TI;                          (* PAWN CONNECTED CREDIT *)
  FPFLNX : TI;                          (* PAWN PHALANX CREDIT *)
  FROUBL : TI;                          (* DOUBLED ROOK CREDIT *)
  FRK7TH : TI;                          (* ROOK ON SEVENTH CREDIT *)
  FTRADE : TI;                          (* TRADE-DOWN BONUS FACTOR *)
  FTRDSL : TI;                          (* TRADE-DOWN TUNING FACTOR *)
  FTRPOK : TI;                          (* PAWN TRADE-DOWN RELAXATION *)
  FTRPWN : TI;                          (* PAWN TRADE-DOWN FACTOR *)
  FWKING : TI;                          (* KING EVALUATION WEIGHT *)
  FWMAJM : TI;                          (* MAJOR PIECE MOBILITY WEIGHT *)
  FWMINM : TI;                          (* MINOR PIECE MOBILITY WEIGHT *)
  FWPAWN : TI;                          (* PAWN EVALUATION WEIGHT *)
  FWROOK : TI;                          (* ROOK EVALUATION WEIGHT *)
  WINDOW : TI;                          (* SIZE OF ALPHA-BETA WINDOW *)


  (* SWITCHES *)

  SWEC : TB;                            (* ECHO INPUT *)
  SWPA : TB;                            (* PAGING *)
  SWPS : TB;                            (* PRINT PRELIMINARY SCORES *)
  SWRE : TB;                            (* REPLY WITH MOVE *)
  SWSU : TB;                            (* PRINT STATISTICS SUMMARY *)
  SWTR : TB;                            (* TRACE TREE SEARCH *)

  (* COMMAND PROCESSING DATA *)

  ICARD : RJ;                           (* INPUT CARD IMAGE *)
  ILINE : RJ;                           (* CURRENT COMMAND *)
  JNTJ  : TJ;                           (* CURRENT INPUT LINE POSITION *)
  JNTJ  : TJ;                           (* CURRENT COMMAND POSITION *)
```

Listing 1, continued:

```
  MOVMS : RN;                          (* MOVE MESSAGE *)

  (* TRANSLATION TABLES *)

  XSPB : ARRAY [TP] OF TB;             (* TRUE FOR SWEEP PIECES *)
  XFPE : ARRAY [TP] OF TE;             (* FIRST DIRECTION *)
  XLLD : ARRAY [AZL..ZAL] OF TD;       (* DIRECTION FOR LARGE BOARD
                                          SQUARE DIFFERENCES *)
  XLPE : ARRAY [TP] OF TE;             (* LAST DIRECTION *)
  XRFS : ARRAY [TF] OF RS;             (* BIT BOARD FOR FILES *)
  XRRS : ARRAY [TR] OF RS;             (* BIT BOARD FOR RANKS *)
  XNFS : ARRAY [TF] OF RS;             (* COMP BIT BOARD FOR FILES *)
  XNRS : ARRAY [TR] OF RS;             (* COMP BIT BOARD FOR RANKS *)
  XRSS : ARRAY [TS] OF RS;             (* BIT BOARD FOR 8X8 INDEX *)
  XRQM : ARRAY [TQ] OF RM;             (* MOVES FOR CASTLE TYPES *)
  XSQS : ARRAY [TQ] OF RS;             (* BIT BOARD FOR CASTLE TYPES *)
  XSSX : ARRAY [TS] OF SX;             (* SET ELEMENT FOR 8X8 INDEX *)
  XTBC : ARRAY [TB] OF TC;             (* CHARACTERS FOR BOOLEANS *)
  XTED : ARRAY [TE] OF TD;             (* DIRECTION NUMBER TO 10X12
                                          SQUARE DIFFERENCE *)
  XTGC : ARRAY [TG] OF TC;             (* CHARACTERS FOR PROMOTION *)
  XTGMP: ARRAY [TG,TM] OF TP;          (* PIECE FOR PROMOTION TYPE
                                          AND COLOR *)
  XTLS : ARRAY [TL] OF TT;             (* 8X8 INDEX FOR 10X12 INDEX *)
  XTMA : ARRAY [TM] OF RA;             (* WORDS FOR COLORS *)
  XTMQ : ARRAY [TM] OF TQ;             (* CASTLE TYPES FOR SIDE *)
  XTMV : ARRAY [TM] OF TV;             (* SCORE FACTOR FOR SIDE *)
  XTPC : ARRAY [TP] OF TC;             (* CHARACTERS FOR PIECES *)
  XTPM : ARRAY [TP] OF TM;             (* SIDES FOR PIECES *)
  XTPU : ARRAY [TP] OF TU;             (* TYPE FOR PIECE *)
  XTPV : ARRAY [TP] OF TV;             (* VALUES OF PIECES *)
  XTQA : ARRAY [TQ] OF RA;             (* WORDS FOR CASTLES *)
  XTQS : ARRAY [TQ] OF TS;             (* TO SQUARES FOR CASTLE TYPES *)
  XTRFS: ARRAY [TR,TF] OF TS;          (* 8X8 INDEX FOR RANK AND FILE *)
  XTSF : ARRAY [TS] OF TF;             (* FILES FOR SQUARES *)
  XTSL : ARRAY [TS] OF TL;             (* 10X12 INDEX FOR 8X8 INDEX *)
  XTSR : ARRAY [TS] OF TR;             (* RANKS FOR SQUARES *)
  XTSX : ARRAY [TS] OF TX;             (* ELEMENT NUMBER FOR 8X8
                                          INDEX *)
  XTSY : ARRAY [TS] OF TY;             (* ARRAY SUBSCRIPT INTO BIT BOARD
                                          FOR 8X8 INDEX *)
  XTUC : ARRAY [TU] OF TC;             (* CHARACTER FOR TYPE *)
  XTUMP: ARRAY [TU,TM] OF TP;          (* PIECE FOR TYPE AND SIDE *)

  XRQSO: ARRAY [TQ] OF RS;             (* UNOCCUPIED SQUARES FOR
                                          CASTLING *)
  XRQSA: ARRAY [TQ] OF RS;             (* UNATTACKED SQUARES FOR
                                          CASTLING *)

  EDGE : ARRAY [TE] OF RS;             (* EDGES IN VARIOUS DIRECTIONS *)
  CORNR: RS;                           (* KING SANCTUARY *)
  NULMV: RM;                           (* NULL MOVE *)
  OTHER: ARRAY [TM] OF TM;             (* OTHER COLOR *)
  SYNTX: ARRAY[SYNCF..SYNML] OF RY;    (* MOVE SYNTAX TABLE *)


FUNCTION MAX(A,B:TI):TI;               (* LARGER OF TWO NUMBERS *)

BEGIN
  IF A > B THEN
    MAX := A
  ELSE
    MAX := B;
END;  (* MAX *)


FUNCTION MIN(A,B:TI):TI;               (* SMALLER OF TWO NUMBERS *)

BEGIN
  IF A < B THEN
    MIN := A
  ELSE
    MIN := B;
END;  (* MIN *)


FUNCTION SIGN(A,B:TI):TI;              (* SIGN OF B APPLIED TO
                                          ABSOLUTE VALUE OF A *)

BEGIN
  SIGN := TRUNC(B/ABS(B)) * ABS(A);
END;  (* SIGN *)


PROCEDURE SORTIT                       (* SORT PRELIMINARY SCORES *)
  (VAR A:RE;                           (* ARRAY OF SCORES *)
   VAR B:RF;                           (* ARRAY OF MOVES *)
   C:TW);                              (* NUMBER OF ENTRIES *)

VAR
  INTB : JB;                           (* LOOP EXIT FLAG *)
  INTW : TW;                           (* OUTER LOOP INDEX *)
  INTI : TI;                           (* INNER LOOP INDEX *)
  INTV : TV;                           (* HOLD SCORE *)
  INRM : RM;                           (* HOLD MOVE *)

BEGIN
  FOR INTW := AW+2 TO C DO
  BEGIN
    INTI := INTW - 1;
    INTV := A[INTW];
    INRM := B[INTW];
    INTB := TRUE;
    WHILE (INTI > AW) AND INTB DO
      IF INTV < A[INTI] THEN
      BEGIN
        A[INTI+1] := A[INTI];
        B[INTI+1] := B[INTI];
        INTI := INTI - 1;
      END
      ELSE
        INTB := FALSE;                 (* EXIT *)
    A[INTI+1] := INTV;
    B[INTI+1] := INRM;
  END;
END;  (* SORTIT *)


PROCEDURE ANDRS                        (* INTERSECTION OF TWO BIT
                                          BOARDS *)
  (VAR C:RS;                           (* RESULT *)
   A, B:RS);                           (* OPERANDS *)

VAR
  INTY : TY;                           (* BIT BOARD WORD INDEX *)

BEGIN
  FOR INTY := AY TO ZY DO
    C.RSSS[INTY] := A.RSSS[INTY] * B.RSSS[INTY];
END;  (* ANDRS *)


PROCEDURE CLRRS                        (* REMOVE SQUARE FROM BIT
                                          BOARD *)
  (VAR C:RS;                           (* BIT BOARD *)
   A:TS);                              (* SQUARE TO REMOVE *)

BEGIN
  C.RSSS[XTSY[A]] := C.RSSS[XTSY[A]] - XSSX[A];
END;  (* CLRRS *)


PROCEDURE CPYRS                        (* COPY OF A BIT BOARD *)
  (VAR C:RS;                           (* RESULT *)
   A:RS);                              (* OPERAND *)

VAR
  INTY : TY;                           (* BIT BOARD WORD INDEX *)

BEGIN
  FOR INTY := AY TO ZY DO
    C.RSSS[INTY] := A.RSSS[INTY];
END;  (* CPYRS *)


PROCEDURE IORRS                        (* UNION OF TWO BIT BOARDS *)
  (VAR C:RS;                           (* RESULT *)
   A, B:RS);                           (* OPERANDS *)

VAR
  INTY : TY;                           (* BIT BOARD WORD INDEX *)

BEGIN
  FOR INTY := AY TO ZY DO
    C.RSSS[INTY] := A.RSSS[INTY] + B.RSSS[INTY];
END;  (* IORRS *)


PROCEDURE NEWRS                        (* CLEAR BIT BOARD *)
  (VAR A:RS);                          (* BIT BOARD TO CLEAR *)

VAR
  INTY : TY;                           (* BIT BOARD WORD INDEX *)

BEGIN
  FOR INTY := AY TO ZY DO
    A.RSSS[INTY] := [];
END;  (* NEWRS *)


PROCEDURE NOTRS                        (* COMPLEMENT OF A BIT BOARD *)
  (VAR C:RS;                           (* RESULT *)
   A:RS);                              (* OPERAND *)

VAR
  INTY : TY;                           (* BIT BOARD WORD INDEX *)

BEGIN
  FOR INTY := AY TO ZY DO
    C.RSSS[INTY] := [AX..ZX]-A.RSSS[INTY];
END;  (* NOTRS *)


FUNCTION NXTTS                         (* NEXT ELEMENT IN BIT BOARD *)
  (VAR A:RS;                           (* BIT BOARD TO LOCATE FIRST
                                          SQUARE, AND THEN REMOVE *)
   VAR B:TS                            (* SQUARE NUMBER OF FIRST SQUARE
                                          IN BIT BOARD *)
   ):TB;                               (* TRUE IFF ANY SQUARES WERE SET
                                          INITIALLY *)

LABEL
  11;                                  (* RETURN *)

VAR
  INTX : TX;                           (* BIT BOARD BIT INDEX *)
  INTY : TY;                           (* BIT BOARD WORD INDEX *)
  X : RK;                              (* KLUDGE WORD *)

BEGIN
  FOR INTY := ZY DOWNTO AY DO          (* LOOP THRU BIT BOARD WORDS *)
    IF A.RSTI[INTY] <> 0 THEN
    BEGIN

(*** BEGIN CDC 6000 DEPENDANT CODE *)
(*** FOLLOWING CODE REQUIRES THE 'EXPO' FUNCTION TO RETURN
(*** THE EXPONENT FROM A FLOATING POINT NUMBER.  IT ALSO ASSUMES
(*** THAT FLOATING POINT NUMBERS HAVE 48 BIT COEFFICIENTS RIGHT-
(*** JUSTIFIED IN A WORD, AND THAT SETS ARE RIGHT-JUSTIFIED IN
(*** A WORD. *)
(*    X.RKTZ := A.RSTI[INTY];          (* FLOAT WORD *)
(*    B := EXPO(X.RKTZ) + INTY * (ZX+1);
(*                                     (* CONVERT TO SQUARE NUMBER *)
```

may not initially be very meaningful, we expect them to be helpful when the user becomes familiar with the program. Because Pascal requires that all procedures and functions be defined in the serial listing before they are called by another portion of the program, the procedures and functions which are first defined tend to be primitives. The main part of the program is concentrated at the end of the listing.

The most important part of the variable declaration list in terms of understanding the program is the portion which specifies the global data base. This includes the current board (BOARD, a record) and a number of important arrays. The look-ahead board (NBORD) is an array listing the piece occupying each square. The attacks emanating from each square are represented by ATKFR, an array which lists an 8 by 8 bit board for each of the 64 squares. The attacks to each square are represented by a similar array, ATKTO. The combined attacks for each side are represented by a 2 item array of 8 by 8 bit boards called ALATK.

The location of all pieces by type is represented by an array of 12 8 by 8 bit boards, TPLOC. The location of all pieces by color is represented by an array of two 8 by 8 bit boards, TMLOC. The moves are stored in an array (MOVES) of records. Each record (RM) contains information about the from square, to square; whether a capture is involved and the type of piece captured, whether the move affects castle status, involves check or mate, involves a piece promotion, and whether the move has been searched yet. Additional arrays provide information on castling squares, en passant squares, the location of all pieces, the location of pawns, etc. To be successful, a chess program must organize the data base in a logical manner and be able to manipulate it efficiently.

For reasons of efficiency, the program often stores the same information in two or more different ways. Because of this, it is necessary to be able to translate from one form to the other. These activities are handled by special arrays. For example, the XTPC array allows one to use a piece designator (LP, LQ, LK, DQ, etc) as an index and returns the corresponding character (1 thru 6 for Black pieces and A thru F for White pieces) which is used when a board representation is printed on the terminal.

There are several general purpose routines which are needed by the program. Two functions, MIN and MAX, provide the smaller or larger of two numbers upon request. A third function, SIGN, applies the sign of one number to the absolute value of another

Listing 1, continued:

```
(*     X.RKTB := X.RKTB - [47];          (* REMOVE MOST SIGNIFICANT BIT *)
(*     A.RSTI[INTY] := TRUNC(X.RKTZ);    (* INTEGERIZE *)
(*     NXTTS := TRUE;                    (* RETURN A BIT SET *)
(*     GOTO 11;                          (* RETURN *)
(*** END CDC 6000 DEPENDANT CODE *)

(*** BEGIN MACHINE INDEPENDENT CODE *)
    FOR INTX := ZX DOWNTO AX DO          (* LOOP THROUGH BITS IN WORD OF
                                            SET *)
      IF INTX IN A.RSSS[INTY] THEN
      BEGIN
        B := INTX+INTY*(ZX+1);           (* RETURN SQUARE NUMBER *)
        A.RSSS[INTY] := A.RSSS[INTY] -
                                         [INTX];
                                         (* REMOVE BIT FROM WORD *)
        NXTTS := TRUE;                   (* RETURN A BIT SET *)
        GOTO 11;                         (* RETURN *)
      END;
(*** END MACHINE INDEPENDENT CODE *)

    END;
  NXTTS := FALSE;                        (* ELSE RETURN NO BITS SET *)
11:  (* RETURN *)
END;  (* NXTTS *)



FUNCTION CNTRS                           (* COUNT MEMBERS OF A BIT
                                            BOARD *)
  (A:RS):TS;                             (* BIT BOARD TO COUNT *)

VAR
  INTY : TY;                             (* BIT BOARD WORD INDEX *)
  INTS : TS;                             (* TEMPORARY *)
  INRS : RS;                             (* SCRATCH *)
  INTS : TS;                             (* SCRATCH *)

BEGIN
  INTS := 0;

(*** BEGIN MACHINE INDEPENDENT CODE *)
  CPYRS(INRS,A);
  WHILE NXTTS(INRS,INTS) DO
    INTS := INTS+1;                      (* COUNT SQUARES *)
(*** END MACHINE INDEPENDENT CODE *)

(*** BEGIN CDC 6000 DEPENDENT CODE *)
(*** FOLLOWING CODE REQUIRES THE 'CARD' FUNCTION TO
(*** COUNT THE MEMBERS IN A SET. *)
(*FOR INTY := AY TO ZY DO
(*  INTS := INTS + CARD(A.RSSS[INTY]);
(*** END CDC DEPENDENT CODE *)

  CNTRS := INTS;                         (* RETURN SUM *)
END;  (* CNTRS *)



PROCEDURE SETRS                          (* INSERT SQUARE INTO BIT
                                            BOARD *)
  (VAR C:RS;                             (* BIT BOARD *)
   A:TS);                                (* SQUARE TO INSERT *)

BEGIN
  C.RSSS[XTSY[A]] := C.RSSS[XTSY[A]] + XSSX[A];
END;  (* SETRS *)



PROCEDURE SFTRS                          (* SHIFT BIT BOARD *)
  (VAR A:RS;                             (* RESULT *)
   B:RS;                                 (* SOURCE *)
   C:TE);                                (* DIRECTION *)

VAR
  INRS : RS;                             (* SCRATCH *)
  INTS : TS;                             (* SCRATCH *)
  INTY : TY;                             (* BIT BOARD WORD INDEX *)

BEGIN

(*** BEGIN MACHINE INDEPENDENT CODE *)
  NEWRS(A);                              (* CLEAR NEW BIT BOARD *)
  WHILE NXTTS(B,INTS) DO
    IF XTLS[XTSL[INTS]+XTED[C]] > 0 THEN
                                         (* SHIFT EACH BIT *)
      SETRS(A,XTLS[XTSL[INTS]+XTED[C]]);
(*** END MACHINE INDEPENDENT CODE *)
(*** BEGIN CDC 6000 DEPENDENT CODE *)
(*** FOLLOWING CODE ASSUMES THAT MULTIPLICATION OR DIVISION
(*** BY A CONSTANT POWER OF 2 IS DONE WITH A SHIFT INSTRUCTION. *)
(*CASE C OF
(*S1:  BEGIN
(*       FOR INTY := AY TO ZY DO         (* SHIFT ONE PLACE *)
(*       BEGIN
(*         B.RSSS[INTY] := B.RSSS[INTY] - EDGE[S1].RSSS[INTY];
(*         A.RSTI[INTY] := B.RSTI[INTY] DIV 2;
(*       END;
(*     END;
(*S2:  BEGIN
(*       FOR INTY := AY TO ZY DO         (* SHIFT WORDS *)
(*       BEGIN
(*         B.RSSS[INTY] := B.RSSS[INTY] - EDGE[S2].RSSS[INTY];
(*         INRS.RSSS[INTY] := B.RSSS[INTY] * [ZX-7..ZX];
(*         A.RSSS[INTY] := B.RSSS[INTY] - [ZX-7..ZX];
(*         A.RSTI[INTY] := A.RSTI[INTY] * 256
(*       END;
(*       FOR INTY := AY+1 TO ZY DO       (* CARRY BETWEEN WORDS *)
(*         A.RSTI[INTY] := A.RSTI[INTY] + INRS.RSTI[INTY-1] DIV PZX8;
(*     END;
(*S3:  BEGIN
(*       FOR INTY := AY TO ZY DO         (* SHIFT ONE PLACE *)
(*       BEGIN
(*         A.RSSS[INTY] := B.RSSS[INTY] - EDGE[S3].RSSS[INTY];
(*         A.RSTI[INTY] := A.RSTI[INTY] * 2;
(*       END;
(*     END;
(*S4:  BEGIN
(*       FOR INTY := AY TO ZY DO         (* SHIFT WORDS *)
(*       BEGIN
(*         B.RSSS[INTY] := B.RSSS[INTY] - EDGE[S4].RSSS[INTY];
(*         INRS.RSSS[INTY] := B.RSSS[INTY] * [AX..AX+7];
(*         A.RSTI[INTY] := B.RSTI[INTY] DIV 256;
(*       END;
(*       FOR INTY := AY TO ZY-1 DO       (* CARRY BETWEEN WORDS *)
(*         A.RSTI[INTY] := A.RSTI[INTY] + INRS.RSTI[INTY+1] * PZX8;
(*     END;
(*B1:  BEGIN
(*       SFTRS(INRS,B,S1);
(*       SFTRS(A,INRS,S2);
(*     END;
(*B2:  BEGIN
(*       SFTRS(INRS,B,S2);
(*       SFTRS(A,INRS,S3);
(*     END;
(*B3:  BEGIN
(*       SFTRS(INRS,B,S3);
(*       SFTRS(A,INRS,S4);
(*     END;
(*B4:  BEGIN
(*       SFTRS(INRS,B,S4);
(*       SFTRS(A,INRS,S1);
(*     END;
(*N1:  BEGIN
(*       SFTRS(INRS,B,B1);
(*       SFTRS(A,INRS,S2);
(*     END;
(*N2:  BEGIN
(*       SFTRS(INRS,B,B2);
(*       SFTRS(A,INRS,S2);
(*     END;
(*N3:  BEGIN
(*       SFTRS(INRS,B,B2);
(*       SFTRS(A,INRS,S3);
(*     END;
(*N4:  BEGIN
(*       SFTRS(INRS,B,B3);
(*       SFTRS(A,INRS,S3);
(*     END;
(*N5:  BEGIN
(*       SFTRS(INRS,B,B3);
(*       SFTRS(A,INRS,S4);
(*     END;
(*N6:  BEGIN
(*       SFTRS(INRS,B,B4);
(*       SFTRS(A,INRS,S4);
(*     END;
(*N7:  BEGIN
(*       SFTRS(INRS,B,B4);
(*       SFTRS(A,INRS,S1);
(*     END;
(*N8:  BEGIN
(*       SFTRS(INRS,B,B1);
(*       SFTRS(A,INRS,S1);
(*     END;
(*END;
(*** END CDC 6000 DEPENDENT CODE *)

END;  (* SFTRS *)



FUNCTION INRSTB                          (* SQUARE IN BIT BOARD BOOLEAN *)
  (A:RS;                                 (* BIT BOARD *)
   B:TS):TB;                             (* SQUARE IN QUESTION *)

BEGIN
  INRSTB := XSSX[B] <= A.RSSS[XTSY[B]];
END;  (* INRSTB *)



FUNCTION NULRS                           (* NULL BIT BOARD *)
  (A:RS)                                 (* BIT BOARD TO CHECK *)
  :TB;                                   (* TRUE IF BIT BOARD EMPTY *)

VAR
  INTY : TY;                             (* BIT BOARD WORD INDEX *)
  INTB : TB;                             (* TEMPORARY VALUE *)

BEGIN
  INTB := TRUE;
  FOR INTY := AY TO ZY DO
    INTB := INTB AND (A.RSTI[INTY] = 0);
  NULRS := INTB;
END;  (* NULRS *)



FUNCTION NULMVB                          (* NULL MOVE BOOLEAN *)
  (A:RM)                                 (* MOVE TO TEST *)
   :TB;                                  (* TRUE IF NULL MOVE *)
BEGIN
  WITH A DO
    NULMVB := RMAC AND RMPR AND (NOT RMCA);
END;  (* NULMVB *)



PROCEDURE INICON;                        (* INITIALIZE GLOBAL CONSTANTS *)

VAR
  INTO : TO;                             (* DIRECTION INDEX *)
  INTE : TE;                             (* DIRECTION *)
  INTF : TF;                             (* FILE INDEX *)
  INTI : TI;                             (* SCRATCH *)
  INTL : TL;                             (* LARGE BOARD INDEX *)
  INTQ : TQ;                             (* CASTLE TYPE INDEX *)
  INTR : TR;                             (* RANK INDEX *)
  INTT : TT;                             (* SQUARE INDEX *)
  INTX : TX;                             (* SET ELEMENT INDEX *)
  INTY : TY;                             (* BIT BOARD WORD INDEX *)
  INTI : TI;                             (* SCRATCH *)
  INRS : RS;                             (* SCRATCH *)
```

number. A general purpose sort routine, SORTIT, is also provided.

Manipulating the Bit Boards

There are a number of primitive operations which involve the manipulation of information represented in bit board form. A bit board is one or more computer words which have a bit set in specific locations to represent the occurrence or nonoccurrence of a particular event. For example eight 8 bit words can be used to represent the eight rows of a chessboard. Each bit corresponds to one square. To represent the location of all White pawns, a bit is set (ie: 1) in the proper locations and all other locations remain clear (ie: 0). This method for representing and manipulating information is very useful in chess programming. For this reason, the first actions defined by our chess program are a set of procedures and functions for manipulating bit boards.

The actions represented are:

(1) the intersection of two bit boards (ANDRS);
(2) the union of two bit boards (IORRS);
(3) the complement of a bit board (NOTRS);
(4) setting a bit in a bit board (SETRS);
(5) removing a bit from a bit board (CLRRS);
(6) counting the number of bits that are set on a bit board (CNTRS);
(7) making a copy of a bit board (CPYRS);
(8) setting all bits to 0 (NEWRS);
(9) shifting all bits in a particular direction (SFTRS);
(10) determining whether a particular bit is set (INRSTB);
(11) determining whether a bit board is empty, ie: has no bits set (NULRS); and
(12) finding and reporting integer value for a location where a bit is set (NXTTS).

Since these routines are used repeatedly by the program, you can decrease the move calculation time quite a bit by implementing these primitives in assembly language. You will note that the function NXTTS is written in two ways: machine independent code, and code which is compatible only with the Control Data 6000 series machines. There are a number of places in the program where execution time can be enhanced by substituting machine dependent code which takes advantage of one or more special features of the hardware you are using. It would be helpful, also, if functions in Pascal could return an array or record instead of just a single value. There are many places in the program where this type of function would be more logical and more efficient than using a procedure (ie: subroutine). If one were to consider the best of all possible worlds, it would be especially nice if the bit map manipulations could be compiled in line. With the Pascal arrangement, many of the procedure calls take as much time as the execution of the procedure.

Initial Steps

It is also necessary at the beginning of the program to provide values for the variables which define the chess environment, such as piece characteristics. For example, a White pawn is represented as LP for some purposes and as the letter A for other purposes. It has the color LITE, is not a sweep piece, and moves only in certain directions. It is necessary to initialize the translation tables, the constant and variable 8 by 8 bit boards, and a number of other tables. The three routines which are called to do this when the program is first activated are INISYN, INIXTP and INICON. A fourth procedure (INITAL) is called by the main program to get ready for a new game. It will be called

Listing 1, continued:

```
  PROCEDURE INISYN                          (* INITIALIZE MOVE SYNTAX
                                               TABLE ENTRY *)
    (A:RA);                                 (* MOVE SYNTAX *)

  BEGIN
    WITH SYNTX[INTI] DO
    BEGIN
      WITH RYLS DO
      BEGIN
        RDPC := TRUE;
        RDSL := A[AA+0] <> " ";
        RDKQ := A[AA+1] <> " ";
        RDN8 := A[AA+2] <> " ";
        RDRK := A[AA+3] <> " ";
      END;
      RYCH := A[AA+4];
      WITH RYRS DO
      BEGIN
        RDPC := A[AA+5] <> " ";
        RDSL := A[AA+6] <> " ";
        RDKQ := A[AA+7] <> " ";
        RDN8 := A[AA+8] <> " ";
        RDRK := A[AA+9] <> " ";
      END;
    END;
    INTI := INTI+1;
  END;  (* INISYN *)


  PROCEDURE INIXTP                          (* INITIALIZE PIECE TRANSLATION
                                               TABLES *)
    (A : TP;                                (* PIECE TO BE TRANSLATED *)
     B : TC;                                (* DISPLAY EQUIVALENT *)
     C : TM;                                (* COLOR OF PIECE *)
     D : TU;                                (* TYPE OF PIECE *)
     E : TB;                                (* TRUE IF SWEEP PIECE *)
     F : TE;                                (* FIRST DIRECTION OF MOVEMENT *)
     G : TE;                                (* LAST DIRECTION OF MOVEMENT *)
     H : TV);                               (* VALUE OF PIECE *)

    BEGIN
      XTPC[A] := B;
      XTPM[A] := C;
      XSPB[A] := E;
      XFPE[A] := F;
      XLPE[A] := G;
      XTPU[A] := D;
      XTPV[A] := H;
      IF A <> MT THEN
        XTUMP[D,C] :=' A;
    END;  (* INIXTP *)


BEGIN  (* INICON *)

  (** INITIALIZE PIECE CHARACTERISTICS *)

  INIXTP(LP,"A",LITE,EP,FALSE,B1,B2,1*64);
  INIXTP(LR,"B",LITE,ER,TRUE ,S1,S4,5*64);
  INIXTP(LN,"C",LITE,EN,FALSE,N1,N8,3*64);
  INIXTP(LB,"D",LITE,EB,TRUE ,B1,B4,3*64);
  INIXTP(LQ,"E",LITE,EQ,TRUE ,B1,S4,9*64);
  INIXTP(LK,"F",LITE,EK,FALSE,B1,S4,0);
  INIXTP(DP,"1",DARK,EP,FALSE,B3,B4,-1*64);
  INIXTP(DR,"2",DARK,ER,TRUE ,S1,S4,-5*64);
  INIXTP(DN,"3",DARK,EN,FALSE,N1,N8,-3*64);
  INIXTP(DB,"4",DARK,EB,TRUE ,B1,B4,-3*64);
  INIXTP(DQ,"5",DARK,EQ,TRUE ,B1,S4,-9*64);
  INIXTP(DK,"6",DARK,EK,FALSE,B1,S4,0);
  INIXTP(MT,"-",NONE,EP,FALSE,B2,B1,0);

  XTGMP[PQ,LITE] := LQ;  XTGMP[PQ,DARK] := DQ;  XTGC[PQ] := "Q";
  XTGMP[PR,LITE] := LR;  XTGMP[PR,DARK] := DR;  XTGC[PR] := "R";
  XTGMP[PN,LITE] := LN;  XTGMP[PN,DARK] := DN;  XTGC[PN] := "N";
  XTGMP[PB,LITE] := LB;  XTGMP[PB,DARK] := DB;  XTGC[PB] := "B";

  XTUC[EK] := "K";
  XTUC[EQ] := "Q";
  XTUC[ER] := "R";
  XTUC[EN] := "N";
  XTUC[EB] := "B";
  XTUC[EP] := "P";

  (** INITIALIZE OTHER CONSTANTS *)

  XTBC[FALSE] := "-";
  XTBC[TRUE ] := "*";

  OTHER[LITE] := DARK;  XTMV[LITE] :=  1;
  OTHER[DARK] := LITE;  XTMV[DARK] := -1;
  OTHER[NONE] := NONE;

  XTMA[LITE] := "    WHITE ";
  XTMA[DARK] := "    BLACK ";
  XTMA[NONE] := "   NO ONE ";

  XTQA[LS] := "WHITE KING";
  XTQA[LL] := "WHITE LONG";
  XTQA[DS] := "BLACK KING";
  XTQA[DL] := "BLACK LONG";

  (** INITIALIZE 10X12 TO 8X8 AND 8X8 TO 10X12 TRANSLATION TABLES *)

  FOR INTL := AL TO ZL DO                   (* LOOP THROUGH LARGE BOARD *)
    XTLS[INTL] := -1;                       (* PRESET ARRAY TO OFF BOARD *)

  INTL := 21;                               (* INDEX OF FIRST SQUARE ON LARGE
                                               BOARD *)
  INTT := -1;                               (* INDEX OF FIRST SQUARE ON SMALL
                                               BOARD *)
  FOR INTR := R1 TO R8 DO                   (* LOOP THROUGH RANKS *)
  BEGIN
    FOR INTF := F1 TO F8 DO                 (* LOOP THROUGH FILES *)
    BEGIN
      INTT := INTT+1;                       (* ADVANCE SMALL BOARD INDEX *)
      XTRFS[INTR,INTF] := INTT;             (* SET MATRIX TO VECTOR
                                               TRANSLATION *)
      XTLS[INTL] := INTT;                   (* SET LARGE BOARD TRANSLATION
                                               TABLE WITH SMALL BOARD
                                               INDEX *)
      XTSL[INTT] := INTL;                   (* SET SMALL BOARD TRANSLATION
                                               TABLE WITH LARGE BOARD
                                               INDEX *)
      XTSR[INTT] := INTR;                   (* SET RANK OF SQUARE *)
      XTSF[INTT] := INTF;                   (* SET FILE OF SQUARE *)
      INTL := INTL+1;                       (* ADVANCE LARGE BOARD INDEX *)
    END;
    INTL := INTL+2;                         (* ADVANCE LARGE BOARD INDEX TO
                                               SKIP BORDER *)
  END;

  (** INITIALIZE 8X8 TO BIT BOARD TABLES *)

  INTT := -1;
  FOR INTY := AY TO ZY DO
  BEGIN
    FOR INTX := AX TO ZX DO
    BEGIN
      INTT := INTT+1;
      XTSX[INTT] := INTX;
      XTSY[INTT] := INTY;
      XSSX[INTT] := [INTX];
      NEWRS(XRSS[INTT]);
      XRSS[INTT].RSSS[INTY] := [INTX];
    END;
  END;

  (** INITIALIZE CONSTANT BIT BOARDS *)

  FOR INTR := R1 TO R8 DO
    NEWRS(XRRS[INTR]);

  FOR INTF := F1 TO F8 DO
    NEWRS(XRFS[INTF]);

  FOR INTR := R1 TO R8 DO
    FOR INTF := F1 TO F8 DO
    BEGIN
      SETRS(XRRS[INTR],XTRFS[INTR,INTF]);
      SETRS(XRFS[INTF],XTRFS[INTR,INTF]);
    END;

  FOR INTF := F1 TO F8 DO
    NOTRS(XNFS[INTF],XRFS[INTF]);

  FOR INTR := R1 TO R8 DO
    NOTRS(XNRS[INTR],XRRS[INTR]);

  (** INITIALIZE EDGES *)

  CPYRS(EDGE[S1],XRFS[F1]);
  CPYRS(EDGE[S2],XRRS[R8]);
  CPYRS(EDGE[S3],XRFS[F8]);
  CPYRS(EDGE[S4],XRRS[R1]);
  IORRS(EDGE[B1],EDGE[S1],EDGE[S2]);
  IORRS(EDGE[B2],EDGE[S2],EDGE[S3]);
  IORRS(EDGE[B3],EDGE[S3],EDGE[S4]);
  IORRS(EDGE[B4],EDGE[S4],EDGE[S1]);
  IORRS(EDGE[N1],EDGE[B1],XRRS[R7]);
  IORRS(EDGE[N2],EDGE[B2],XRRS[R7]);
  IORRS(EDGE[N3],EDGE[B2],XRFS[F7]);
  IORRS(EDGE[N4],EDGE[B3],XRFS[F7]);
  IORRS(EDGE[N5],EDGE[B3],XRRS[R2]);
  IORRS(EDGE[N6],EDGE[B4],XRRS[R2]);
  IORRS(EDGE[N7],EDGE[B4],XRFS[F2]);
  IORRS(EDGE[N8],EDGE[B1],XRFS[F2]);

  (** INITIALIZE CORNER MASK *)

  IORRS(INRS,XRRS[R1],XRRS[R2]);
  IORRS(INRS,INRS,XRRS[R7]);
  IORRS(INRS,INRS,XRRS[R8]);
  IORRS(CORNR,XRFS[F1],XRFS[F2]);
  IORRS(CORNR,CORNR,XRFS[F7]);
  IORRS(CORNR,CORNR,XRFS[F8]);
  ANDRS(CORNR,CORNR,INRS);

  (** INITIALIZE DIRECTION TABLE *)

                XTED[N1]:= 19;                XTED[N2]:= 21;
  XTED[N8]:=  8;XTED[B1]:=  9;XTED[S2]:= 10;XTED[B2]:= 11;XTED[N3]:= 12;
                XTED[S1]:= -1;                XTED[S3]:=  1;
  XTED[N7]:=-12;XTED[B4]:=-11;XTED[S4]:=-10;XTED[B3]:= -9;XTED[N4]:= -8;
                XTED[N6]:=-21;                XTED[N5]:=-19;

  (** INITIALIZE SQUARE DIFFERENCE TO DIRECTION TABLE *)

  FOR INTI := AZL TO ZAL DO
    XLLD[INTI] := 0;
  FOR INTE := B1 TO S4 DO
  BEGIN
    INTD := XTED[INTE];
    FOR IMTI := 1 TO 7 DO
      XLLD[IMTI*INTD] := INTD;
  END;
  FOR INTE := N1 TO N8 DO
    XLLD[XTED[INTE]] := XTED[INTE];

  (** INITIALIZE CASTLING TRANSLATION TABLES *)

  IORRS(XSQS[LS],XRSS[XTRFS[R1,F8]],XRSS[XTRFS[R1,F5]]);
  IORRS(XSQS[LL],XRSS[XTRFS[R1,F1]],XRSS[XTRFS[R1,F5]]);
  IORRS(XSQS[DS],XRSS[XTRFS[R8,F8]],XRSS[XTRFS[R8,F5]]);
  IORRS(XSQS[DL],XRSS[XTRFS[R8,F1]],XRSS[XTRFS[R8,F5]]);

  IORRS(XRQSO[LS],XRSS[XTRFS[R1,F6]],XRSS[XTRFS[R1,F7]]);
  IORRS(XRQSO[LL],XRSS[XTRFS[R1,F4]],XRSS[XTRFS[R1,F3]]);
  IORRS(XRQSA[LS],XRSS[XTRFS[R1,F5]],XRQSO[LS]);
  IORRS(XRQSA[LL],XRSS[XTRFS[R1,F5]],XRQSO[LL]);
  IORRS(XRQSO[LL],XRSS[XTRFS[R1,F2]],XRQSO[LL]);

  IORRS(XRQSO[DS],XRSS[XTRFS[R8,F6]],XRSS[XTRFS[R8,F7]]);
  IORRS(XRQSO[DL],XRSS[XTRFS[R8,F4]],XRSS[XTRFS[R8,F3]]);
```

more than once if the user wishes to play more than one game.

During the development of the program, it is necessary to determine whether the individual procedures are functioning properly. To do this, it is helpful to have a few primitive print routines which can provide information about the internal workings in a form which is understandable to the programmer. These same routines are also called by the main input/output (IO) routine (READER) which appears later in the program.

One of these routines (PRIMOV) prints an internal representation of the machine's move. Another prints an 8 by 8 array representing the board (PRINTB). This consists of numbers for Black's pieces (Black pawn = 1; Black King = 6) and letters for White's pieces (White pawn = A; White King = F) with empty squares represented by a —. The PRINBB routine prints an 8 by 8 array representing a bit board. In this case an asterisk (*) stands for a square where a bit is set and a minus sign (–) stands for a square where a bit has not been set. An attack map is printed by PRINAM and this consists of 64 (one for each square) 8 by 8 bit maps in which an * stands for a bit which is set and a – stands for a clear bit.

Other useful print routines include one which permits a user controlled pause during printing (PAUSER) and one which informs the programmer of the status of particular control switches (PRISWI). Because of Pascal's serial requirement (ie: every procedure must be defined before it can be called by another procedure), these routines appear early in the program so that they can be used to test the procedures and functions which follow.

In part 1 we mentioned incremental updating as an important feature of an efficient chess program. It is necessary to apply an evaluation function to the terminal nodes of the look-ahead tree. These evaluations, if they are at all sophisticated, require a substantial amount of detailed information about the position. Although it is possible to calculate this information separately for each evaluation, this is not a very efficient procedure, because adjacent nodes are almost identical. Most of the information which would be calculated each time would be redundant. A more efficient alternative is to "update" and "downdate" the relevant data base incrementally as the program moves about in the look-ahead tree. This capability requires quite a bit of special programming.

Several primitive routines are very useful for this. If the move involves a capture, it

Listing 1, continued:

```
IORRS(XRQSA[DS],XRSS[XTRFS[R8,F5]],XRQSO[DS]);
IORRS(XRQSA[DL],XRSS[XTRFS[R8,F5]],XRQSO[DL]);
IORRS(XRQSO[DL],XRSS[XTRFS[R8,F2]],XRQSO[DL]);

FOR INTQ := LS TO DL DO
  WITH XRQM[INTQ] DO
  BEGIN
    RMCP := MT;
    RMCA := FALSE;
    RMAC := TRUE;
    RMCH := FALSE;
    RMMT := FALSE;
    RMIL := FALSE;
    RMSU := FALSE;
    RMPR := FALSE;
    RMOO := TRUE;
  END;

XRQM[LS].RMFR := XTRFS[R1,F5]; XRQM[LS].RMTO := XTRFS[R1,F7];
XRQM[LL].RMFR := XTRFS[R1,F5]; XRQM[LL].RMTO := XTRFS[R1,F3];
XRQM[DS].RMFR := XTRFS[R8,F5]; XRQM[DS].RMTO := XTRFS[R8,F7];
XRQM[DL].RMFR := XTRFS[R8,F5]; XRQM[DL].RMTO := XTRFS[R8,F3];

XRQM[LS].RMQS := FALSE;
XRQM[LL].RMQS := TRUE;
XRQM[DS].RMQS := FALSE;
XRQM[DL].RMQS := TRUE;

XTMQ[LITE] := LS;
XTMQ[DARK] := DS;

XTQS[LS] := XTRFS[R1,F8];
XTQS[LL] := XTRFS[R1,F1];
XTQS[DS] := XTRFS[R8,F8];
XTQS[DL] := XTRFS[R8,F1];

(** INITIALIZE NULL MOVE *)

WITH NULMV DO
BEGIN
  RMFR := AS;
  RMTO := AS;
  RMCP := MT;
  RMCA := FALSE;
  RMAC := TRUE;
  RMCH := FALSE;
  RMMT := FALSE;
  RMIL := FALSE;
  RMSU := FALSE;
  RMPR := TRUE;
  RMPP := PB;
END;

(** INITIALIZE COMMAND PROCESSING VARIABLES *)

JMTJ := ZJ;
ICARD[ZJ] := ";";
ILINE[ZJ] := ";";

(** INITIALIZE MOVES SYNTAX TABLE *)

INTI := SYNCF;
INISYN("    *P    ");
INISYN("    *P/  1");
INISYN("/  1*P    ");
INISYN("    *P/ R ");
INISYN("/ R *P    ");
INISYN("    *P/ R1");
INISYN("/ R1*P    ");
INISYN("    *P/KR ");
INISYN("/KR *P    ");
INISYN("    *P/KR1");
INISYN("/KR1*P    ");
INISYN("/  1*P/  1");
INISYN("/ R *P/ R ");
INISYN("/  1*P/ R ");
INISYN("/ R *P/  1");
INISYN("/ R1*P/  1");
INISYN("/  1*P/ R1");
INISYN("/ R1*P/ R ");
INISYN("/ R *P/ R1");
INISYN("/KR *P/  1");
INISYN("/  1*P/KR ");
INISYN("/KR *P/ R ");
INISYN("/ R *P/KR ");
INISYN("/  1*P/KR1");
INISYN("/KR1*P/  1");
INISYN("  R *P/KR1");
INISYN("/KR1*P/ R ");
INISYN("/ R1*P/ R1");
INISYN("/KR *P/ R1");
INISYN("/ R1*P/KR ");
INISYN("/KR *P/KR ");
INISYN("/KR1*P/ R1");
INISYN("/ R1*P/KR1");
INISYN("/KR1*P/KR ");
INISYN("/KR *P/KR1");
INISYN("/KR1*P/KR1");

INISYN("    -   R1");
INISYN("    -  KR1");
INISYN("/  1-   R1");
INISYN("/ R -   R1");
INISYN("/  1-  KR1");
INISYN("/ R -  KR1");
INISYN("/ R1-   R1");
INISYN("/KR -   R1");
INISYN("/ R1-  KR1");
INISYN("/KR -  KR1");
INISYN("/KR1-  KR1");

(** INITIALIZE LETS *)

FKPSHD := 10;
FKSANQ := 150;
FMAXMT := 256;
FNODEL := 18;
FPADCR[F1] := 0;
FPADCR[F2] := 0;
FPADCR[F3] := 5;
FPADCR[F4] := 10;
FPADCR[F5] := 15;
FPADCR[F6] := 5;
FPADCR[F7] := 0;
FPADCR[F8] := 0;
FPBLOK := 20;
FPCONN := 5;
FPFLNX := 12;
FRDUBL := 60;
FRK7TH := 120;
FTRADE := 36;
FTRDSL := 5156;
FTRPOK := 2;
FTRPWN := 8;
FWKING := 50;
FWMAJM := 1;
FWMINM := 200;
FWPAWN := 100;
FWROOK := 2;
WINDOW := 30;

(** INITIALIZE SWITCHES *)

SWEC := TRUE;
SWPA := TRUE;
SWPS := FALSE;
SWRE := TRUE;
SWSU := FALSE;
SWTR := FALSE;

(** INITIALIZE MAIN LOOP CONTROL VARIABLES *)

GOING := 0;

END;  (* INICON *)


PROCEDURE INITAL(VAR A:RB);              (* INITIALIZE FOR A NEW GAME *)

VAR
  INTF : TF;                             (* FILE INDEX *)
  INTR : TR;                             (* RANK INDEX *)

BEGIN
  WITH A DO
  BEGIN
    RBTM := LITE;                        (* SIDE TO MOVE *)
    RBTS := -1;                          (* NO ENPASSANT SQUARE *)
    RBTI := 0;                           (* GAME HAS NOT STARTED *)
    RBSQ := [LS,LL,DS,DL];               (* ALL CASTLING MOVES LEGAL *)
    FOR INTF := F1 TO F8 DO              (* LOOP THROUGH ALL FILES *)
    BEGIN
      RBIRF[R2,INTF] := LP;              (* SET LIGHT PAWNS ON BOARD *)
      FOR INTR := R3 TO R6 DO            (* LOOP THRU MIDDLE OF BOARD *)
        RBIRF[INTR,INTF] := MT;          (* SET MIDDLE OF BOARD EMPTY *)
      RBIRF[R7,INTF] := DP;              (* SET DARK PAWNS ON BOARD *)
    END;
    RBIRF[R1,F1] := LR;                  (* SET REMAINDER OF PIECES ON
                                            BOARD *)
    RBIRF[R1,F2] := LN;
    RBIRF[R1,F3] := LB;
    RBIRF[R1,F4] := LQ;
    RBIRF[R1,F5] := LK;
    RBIRF[R1,F6] := LB;
    RBIRF[R1,F7] := LN;
    RBIRF[R1,F8] := LR;
    RBIRF[R8,F1] := DR;
    RBIRF[R8,F2] := DN;
    RBIRF[R8,F3] := DB;
    RBIRF[R8,F4] := DQ;
    RBIRF[R8,F5] := DK;
    RBIRF[R8,F6] := DB;
    RBIRF[R8,F7] := DN;
    RBIRF[R8,F8] := DR;

    MOVMS := " ENTER MOVE OR TYPE GO.        ";
    WRITELN(MOVMS);
    LSTMV := NULMV;                      (* INITIALIZE PREVIOUS MOVE *)
  END;
END;  (* INITAL *)


PROCEDURE PAUSER;                        (* PAUSE FOR CARRIAGE RETURN *)

BEGIN
  IF SWPA THEN
  BEGIN
    WRITELN(" PAUSING ");
    READLN;
  END;
END;  (* PAUSER *)


PROCEDURE PRIMOV(A:RM);                  (* PRINT A MOVE *)

BEGIN
  WITH A DO
  BEGIN
    WRITE(" FROM ",RMFR:2," TO ",RMTO:2);
    IF NULMVB(A) THEN
      WRITE(", NULL MOVE")
    ELSE
    BEGIN
      IF RMCA THEN
        WRITE(", CAPTURE ",XTPC[RMCP],",")
      ELSE
        WRITE(", SIMPLE,");
      IF NOT RMAC THEN
        WRITE(" NO");
      WRITE(" ACS");
      IF RMCH THEN
```

is necessary to change the material balance function. The actual scoring itself is handled by MBEVAL. This routine is called either by MBCAPT or MBTPAC when a piece is lost (update) or gained (downdate); or by MBPROM or MBMORP when a pawn is promoted (update); or when a newly promoted pawn is demoted (downdate). There are other changes which are required in the data base for both capture and noncapture moves. The new squares which are attacked by the piece need to be added to the attack maps (ATKFR, ATKTO, ALATK). This is done by ADDATK. The new square for the piece is added to the data base by ADDLOC. The attacks of sliding pieces which are blocked by the newly moved piece are recomputed by CUTATK. The attacks of sliding pieces which are unblocked by vacating the former square are recomputed by PRPATK. The attacks which emanated from the piece on its former square are deleted by DELATK. These primitive routines are called by LOSEIT when a capture is involved or by MOVEIT otherwise. If the move affects castling status, the necessary data base changes are made by PROACA and PROACS. If a pawn promotion is involved, PROMOT makes the necessary adjustments.

Move Generation

A major part of any chess program is the move generation module. Because of the complexity of the game, many programs simply ignore some of the more unusual moves, such as Queenside castling, en passant pawn captures, or promotion of a pawn to a piece other than a Queen (ie: underpromotion). This arrangement will suffice to play legal chess, but it may be costly if one of the omitted move types is highly desirable in a specific game situation. In addition, an incomplete move generation facility prevents the machine from checking the legality of its opponent's moves.

Rather than being satisfied with an approximate solution, we have heeded the old maxim, "If a job is worth doing, it is worth doing well," and have implemented a move generator which permits the program to play a complete game of legal chess. As you can see from the listing, this requires extensive programming.

The first step in move generation is to create the data base for the important features of the existing board configuration. This is done by CREATE. Once a move has been selected, it is necessary to change the data base. This is done by UPDATE which

Text continued on page 181

Listing 1, continued:

```
        WRITE(', CHECK');
      IF RMMT THEN
        WRITE(', MATE');
      IF RMIL THEN
        WRITE(', ILLEGAL');
      IF RMSU THEN
        WRITE(', SEARCHED');
      CASE RMPR OF
      FALSE:  (* NOT PROMOTION *)
        CASE RMOO OF
        FALSE:  (* NOT CASTLE *)
          IF RMEP THEN
            WRITE(', ENPASSANT');
        TRUE:  (* CASTLE *)
          BEGIN
            WRITE(', CASTLE ');
            IF RMQS THEN
              WRITE('LONG')
            ELSE
              WRITE('SHORT');
          END;
        END;
      TRUE:   (* PROMOTION *)
        BEGIN
          WRITE(', PROMOTE TO ');
          CASE RMPP OF
          PQ:  WRITE('QUEEN');
          PR:  WRITE('ROOK');
          PB:  WRITE('BISHOP');
          PN:  WRITE('KNIGHT');
          END;
        END;
      END;
    END;
  END;
  WRITELN('.');
END;  (* PRIMOV *)


PROCEDURE PRINTB(A:RC);                    (* PRINT A BOARD *)

VAR
  INTR : TR;                               (* RANK INDEX *)
  INTF : TF;                               (* FILE INDEX *)

BEGIN
  WRITELN;                                 (* WRITE A BLANK LINE *)
  FOR INTR := R8 DOWNTO R1 DO              (* LOOP DOWN THROUGH RANKS *)
  BEGIN
    WRITE (" ",ORD(INTR)+1:1," ");         (* OUTPUT RANK LABEL *)
    FOR INTF := F1 TO F8 DO                (* LOOP ACROSS THROUGH FILES *)
      WRITE (XTPC[A[XTRFS[INTR,INTF]]]);
                                           (* OUTPUT CONTENTS OF SQUARE *)
    WRITELN;                               (* WRITE OUT A RANK *)
  END;
  WRITELN (" W RNBQKBNR");                 (* WRITE OUT BOTTOM LABEL *)
END;  (* PRINTB *)


PROCEDURE PRINBB(A:FS);                    (* PRINT A BIT BOARD *)

VAR
  INTR : TR;                               (* RANK INDEX *)
  INTF : TF;                               (* FILE INDEX *)

BEGIN
  WRITELN;                                 (* WRITE OUT A BLANK LINE *)
  FOR INTR := R8 DOWNTO R1 DO              (* LOOP DOWN THROUGH RANKS *)
  BEGIN
    WRITE (" ",ORD(INTR)+1:1," ");         (* OUTPUT RANK LABEL *)
    FOR INTF := F1 TO F8 DO                (* LOOP ACROSS THROUGH FILES *)
      WRITE (XTBC[INRSTB(A,XTRFS[INTR,INTF])]);
                                           (* OUTPUT CONTENTS OF SQUARE *)
    WRITELN;                               (* WRITE OUT A RANK *)
  END;
  WRITELN (" W RNBQKBNR");                 (* WRITE OUT BOTTOM LABEL *)
END;  (* PRINBB *)


PROCEDURE PRINAM(A:FX);                    (* PRINT ATTACK MAP *)

VAR
  INTR, JNTR : TR;                         (* RANK INDICES *)
  INTF, JNTF : TF;                         (* FILE INDICES *)

BEGIN
  WRITELN;
  FOR INTR := R8 DOWNTO R1 DO
  BEGIN
    FOR JNTR := R8 DOWNTO R1 DO
    BEGIN
      FOR INTF := F1 TO F8 DO
      BEGIN
        WRITE(" ");
        FOR JNTF := F1 TO F8 DO
        BEGIN
          WRITE(XTBC[INRSTB(A[XTRFS[INTR,INTF]],XTRFS[JNTR,JNTF])]);
        END;
        WRITE(" ");
      END;
      WRITELN;
    END;
    WRITELN;
    IF INTR IN [R1,R3,R5,R7] THEN PAUSER;
  END;
END;  (* PRINAM *)


PROCEDURE PRISWI(A:FA;B:TB);               (* PRINT A SWITCH *)

BEGIN
  WRITE(" ",A[AA],A[AA+1]);
  IF B THEN
    WRITELN(" ON")
  ELSE
    WRITELN(" OFF");
END;  (* PRISWI *)


PROCEDURE MBEVAL;                          (* EVALUATE MATERIAL BALANCE *)

VAR
  INTI : TI;                               (* COUNT PAWNS OF WINNING SIDE *)

BEGIN
  IF MBLTE <> 0 THEN
    IF MBLTE > 0 THEN
      INTI := MBPWN[LITE]
    ELSE
      INTI := MBPWN[DARK]
  ELSE
    INTI := 0;

  MBVAL[JNTK] := SIGN(MIN(MIN(FMAXMT,ABS(MBLTE))
      +FTRADE*ABS(MBLTE)*(FTRDSL-MBTOT)*(4*INTI+FTRPOK)
      DIV (4*INTI+FTRPWN) DIV 262144,16320),MBLTE);

END;  (* MBEVAL *)


PROCEDURE MBCAPT                           (* EVALUATE MATERIAL AFTER
                                              CAPTURE *)
  (A:TP);                                  (* PIECE CAPTURED *)

BEGIN
  MBTOT := MBTOT - ABS(XTPV[A]);           (* TOTAL MATERIAL ON BOARD *)
  IF XTPU[A] = EP THEN
    MBPWN[XTPM[A]] := MBPWN[XTPM[A]] - 1;
                                           (* REMOVE PAWN IF NECESSARY *)
  MBLTE := MBLTE - XTPV[A];                (* LITE ADVANTAGE *)
  MBEVAL;                                  (* EVALUATE MATERIAL *)
END;  (* MBCAPT *)


PROCEDURE MBTPAC                           (* REMOVE CAPTURE FROM
                                              MATERIAL BALANCE DATA.  THIS
                                              IS THE INVERSE OF MBCAPT *)
  (A:TP);                                  (* PIECE UNCAPTURED *)

BEGIN
  MBTOT := MBTOT + ABS(XTPV[A]);
  IF XTPU[A] = EP THEN
    MBPWN[XTPM[A]] := MBPWN[XTPM[A]] + 1;
  MBLTE := MBLTE + XTPV[A];
END;  (* MBTPAC *)


PROCEDURE MBPROM                           (* EVALUATE MATERIAL BALANCE
                                              CHANGE DUE TO PAWN
                                              PROMOTION *)
  (A:TP);                                  (* PIECE TO PROMOTE TO *)

BEGIN
  MBTOT := MBTOT + ABS(XTPV[A]-XTPV[XTUMP[EP,XTPM[A]]]);
                                           (* TOTAL MATERIAL ON BOARD *)
  MBPWN[XTPM[A]] := MBPWN[XTPM[A]] - 1;(* COUNT PAWNS *)
  MBLTE := MBLTE + XTPV[A]-XTPV[XTUMP[EP,XTPM[A]]];
  MBEVAL;                                  (* EVALUATE RESULT *)
END;  (* MBPROM *)


PROCEDURE MBMORP                           (* REMOVE PAWN PROMOTION
                                              FROM MATERIAL BALANCE DATA.
                                              THIS IS THE INVERSE
                                              OF MBPROM *)
  (A:TP);                                  (* PIECE PROMOTED TO *)

BEGIN
  MBTOT := MBTOT - ABS(XTPV[A]-XTPV[XTUMP[EP,XTPM[A]]]);
  MBPWN[XTPM[A]] := MBPWN[XTPM[A]] + 1;
  MBLTE := MBLTE - (XTPV[A]-XTPV[XTUMP[EP,XTPM[A]]]);
END;  (* MBMORP *)


PROCEDURE ADDATK                           (* ADD ATTACKS OF PIECE TO DATA
                                              BASE *)
  (A:TS);                                  (* SQUARE OF PIECE TO ADD
                                              ATTACK *)

VAR
  INTB : TB;                               (* LOOP CONTROL BOOLEAN *)
  INTO : TO;                               (* CURRENT DIRECTION OFFSET *)
  INTE : TE;                               (* CURRENT DIRECTION INDEX *)
  INTM : TM;                               (* COLOR OF CURRENT PIECE *)
  INTP : TP;                               (* CURRENT PIECE *)
  INTT : TT;                               (* RUNNING SQUARE *)

BEGIN
  INTP := NBORD[A];                        (* PIECE OF INTEREST *)
  INTM := XTPM[INTP];                      (* COLOR *)
  FOR INTE := XFPE[INTP] TO XLPE[INTP] DO
  BEGIN
    INTT := A;                             (* INITIALIZE RUNNING SQUARE *)
    INTB := XSPB[INTP];                    (* TRUE IF SWEEP PIECE *)
    INTO := XTED[INTE];                    (* OFFSET *)
    REPEAT
      INTT := XTLS[XTSL[INTT] + INTO];     (* STEP IN PROPER DIRECTION *)
      IF INTT >= 0 THEN
      BEGIN
        SETRS(ATKFR[A],INTT);
        SETRS(ATKTO[INTT],A);
        SETRS(ALATK[INTM],INTT);
        IF NBORD[INTT] <> MT THEN
          INTB := FALSE;
```

Listing 1, continued:

```
      END
      ELSE
        INTB := FALSE;
    UNTIL NOT INTB;
  END;
END;  (* ADDATK *)


PROCEDURE ADDLOC                        (* ADD PIECE TO DATA BASE *)
  (A:TS;                                (* SQUARE WITH NEW PIECE ON IT *)
   B:TP);                               (* NEW PIECE TO ADD *)

BEGIN
  CLRRS(TPLOC[MT],A);                   (* BIT BOARD OF EMPTY SQUARES *)
  SETRS(TPLOC[B],A);                    (* BIT BOARD OF ALL SAME PIECE *)
  SETRS(TMLOC[XTPM[B]],A);              (* BIT BOARD OF ALL SAME COLOR *)
  SETRS(ALLOC[JMTK],A);                 (* BIT BOARD OF ALL PIECES *)
  NBORD[A] := B;                        (* SET NEW PIECE ON BOARD *)
END;  (* ADDLOC *)


PROCEDURE CLSTAT;                       (* CLEAR POSITION STATUS *)

BEGIN
  WITH BOARD DO
  BEGIN
    RBTM := LITE;                       (* WHITE TO MOVE *)
    RBTS := -1;                         (* NO ENPASSANT *)
    RBSQ := [];                         (* NO CASTLING LEGAL *)
  END;
END;  (* CLSTAT *)


PROCEDURE CUTATK                        (* CUT ATTACKS THROUGH SQUARE *)
  (A:TS);                               (* SQUARE *)

VAR
  INRS : RS;                            (* ATTACKING PIECES *)
  INTS : TS;                            (* ATTACKING PIECE SQUARE *)
  IMRS : RS;                            (* SCRATCH *)
  INTD : TD;                            (* STEP SIZE *)
  INTM : TM;                            (* ATTACKING PIECE SIDE *)
  INTL : TL;                            (* NO LONGER ATTACKED SQUARE *)
  INTT : TT;                            (* NO LONGER ATTACKED SQUARE *)

BEGIN
  CPYRS(INRS,ATKTO[A]);                 (* ALL PIECES ATTACKING SQUARE *)
  WHILE NXTTS(INRS,INTS) DO
    IF XSPB[NBORD[INTS]] THEN           (* IF SWEEP PIECE *)
    BEGIN
      INTD := XLLD[XTSL[A]-XTSL[INTS]];
                                        (* STEP SIZE ON 10 X 12 BOARD *)
      INTM := XTPM[NBORD[INTS]];        (* SIDE OF ATTACKING PIECE *)
      INTL := XTSL[A]+INTD;             (* FIRST SQUARE BEYOND PIECE *)
      INTT := XTLS[INTL];               (* FIRST SQUARE BEYOND PIECE ON
                                           8X8 BOARD *)
      WHILE INTT > AT DO                (* WHILE ON BOARD *)
      BEGIN
        CLRRS(ATKFR[INTS],INTT);        (* CLEAR ATTACK MAP *)
        CLRRS(ATKTO[INTT],INTS);
        ANDRS(IMRS,ATKTO[INTT],TMLOC[INTM]);
                                        (* OTHER ATTACKS ON SQUARE BY
                                           SAME SIDE *)
        IF NULRS(IMRS) THEN             (* IF NO ATTACKS BY THAT SIDE *)
          CLRRS(ALATK[INTM],INTT);      (* CLEAR ATTACKS BY SIDE *)
        IF NBORD[INTT] = MT THEN
        BEGIN
          INTL := INTL+INTD;            (* STEP BEYOND SQUARE *)
          INTT := XTLS[INTL];
        END
        ELSE
          INTT := AT;                   (* STOP SCAN *)
      END;
    END;
  END;
END;  (* CUTATK *)


PROCEDURE DELATK                        (* DELETE ATTACKS FROM SQUARE *)
  (A:TS);                               (* SQUARE TO REMOVE PIECE *)

VAR
  INRS : RS;                            (* SQUARES ATTACKED BY PIECE ON
                                           SQUARE *)
  IMRS : RS;                            (* SCRATCH *)
  INTS : TS;                            (* SQUARE ATTACKED BY PIECE ON
                                           SQUARE *)
  INTM : TM;                            (* SIDE OF PIECE ON SQUARE *)

BEGIN
  CPYRS(INRS,ATKFR[A]);                 (* SQUARES ATTACKED BY PIECE
                                           ON SQUARE *)
  NEWRS(ATKFR[A]);                      (* CLEAR ATTACKS FROM SQUARE *)
  INTM := XTPM[NBORD[A]];               (* SIDE OF PIECE ON SQUARE *)
  WHILE NXTTS(INRS,INTS) DO             (* LOOP THROUGH ALL ATTACKS BY
                                           PIECE *)
  BEGIN
    CLRRS(ATKTO[INTS],A);               (* CLEAR ATTACK TO OTHER
                                           SQUARE *)
    ANDRS(IMRS,ATKTO[INTS],TMLOC[INTM]);
                                        (* OTHER ATTACKS BY SAME SIDE *)
    IF NULRS(IMRS) THEN
      CLRRS(ALATK[INTM],INTS);          (* CLEAR ATTACKS BY SIDE *)
    CLRRS(TPLOC[NBORD[A]],A);           (* CLEAR PIECE *)
    CLRRS(TMLOC[INTM],A);               (* CLEAR PIECE FROM SIDE *)
    CLRRS(ALLOC[JMTK],A);               (* CLEAR PIECE FROM ALL PIECES *)
    SETRS(TPLOC[MT],A);                 (* SET EMPTY *)
    NBORD[A] := MT;
  END;
END;  (* DELATK *)
```

```
PROCEDURE PRPATK                        (* PROPAGATE ATTACKS THROUGH
                                           SQUARE *)
  (A:TS);                               (* SQUARE *)

VAR
  INRS : RS;                            (* ATTACKING PIECES *)
  INTS : TS;                            (* ATTACKING PIECE SQUARE *)
  INTD : TD;                            (* STEP SIZE *)
  INTM : TM;                            (* ATTACKING PIECE SIDE *)
  INTL : TL;                            (* NEW ATTACKED SQUARE *)
  INTT : TT;                            (* NEW ATTACKED SQUARE *)

BEGIN
  CPYRS(INRS,ATKTO[A]);                 (* ALL PIECES ATTACKING SQUARE *)
  WHILE NXTTS(INRS,INTS) DO
    IF XSPB[NBORD[INTS]] THEN           (* IF SWEEP PIECE *)
    BEGIN
      INTD := XLLD[XTSL[A]-XTSL[INTS]];
                                        (* STEP SIZE ON 10 X 12 BOARD *)
      INTM := XTPM[NBORD[INTS]];        (* SIDE OF ATTACKING PIECE *)
      INTL := XTSL[A]+INTD;             (* FIRST SQUARE BEYOND PIECE *)
      INTT := XTLS[INTL];               (* FIRST SQUARE BEYOND PIECE ON
                                           8X8 BOARD *)
      WHILE INTT >= 0 DO                (* WHILE ON BOARD *)
      BEGIN
        SETRS(ATKFR[INTS],INTT);        (* SET ATTACK MAP *)
        SETRS(ATKTO[INTT],INTS);
        SETRS(ALATK[INTM],INTT);        (* SET ATTACKS BY SIDE *)
        IF NBORD[INTT] = MT THEN
        BEGIN
          INTL := INTL+INTD;            (* STEP BEYOND SQUARE *)
          INTT := XTLS[INTL];
        END
        ELSE
          INTT := -1;                   (* STOP SCAN *)
      END;
    END;
END;  (* PRPATK *)


PROCEDURE GAINIT                        (* UNPROCESS CAPTURE MOVE *)
  (A:RM);                               (* CAPTURE MOVE *)

BEGIN
  WITH A DO
  BEGIN
    ADDLOC(RMFR,NBORD[RMTO]);           (* PUT PIECE ON ORIGINAL
                                           SQUARE *)
    ADDATK(RMFR);
    CUTATK(RMFR);                       (* STOP ATTACKS AT THIS SQUARE *)
    DELATK(RMTO);                       (* REMOVE THEM FROM
                                           DESTINATION SQUARE *)
    ADDLOC(RMTO,RMCP);                  (* REPLACE CAPTURED PIECE *)
    ADDATK(RMTO);
    MBTPAC(NBORD[RMTO]);                (* UPDATE SCORE *)
  END;
END;  (* GAINIT *)


PROCEDURE LOSEIT                        (* PROCESS CAPTURE MOVE *)
  (A:RM);                               (* CAPTURE MOVE *)

BEGIN
  WITH A DO
  BEGIN
    MBCAPT(NBORD[RMTO]);                (* UPDATE SCORE *)
    DELATK(RMTO);                       (* DELETE ATTACKS OF CAPTURED
                                           PIECE *)
    ADDLOC(RMTO,NBORD[RMFR]);           (* ADD PIECE TO DESTINATION
                                           SQUARE *)
    DELATK(RMFR);                       (* DELETE ATTACKS OF MOVING
                                           PIECE *)
    PRPATK(RMFR);                       (* PROPAGATE ATTACKS THROUGH
                                           FROM SQUARE *)
    ADDATK(RMTO);                       (* ADD ATTACKS OF MOVING PIECE *)
  END;
END;  (* LOSEIT *)


PROCEDURE MOVEIT                        (* PROCESS ORDINARY MOVE *)
  (A:RM);                               (* ORDINARY MOVE *)

BEGIN
  WITH A DO
  BEGIN
    ADDLOC(RMTO,NBORD[RMFR]);           (* ADD PIECE TO NEW SQUARE *)
    CUTATK(RMTO);                       (* CUT ATTACKS THROUGH NEW
                                           SQUARE *)
    DELATK(RMFR);                       (* DELETE ATTACKS FROM OLD
                                           SQUARE *)
    PRPATK(RMFR);                       (* PROPAGATE ATTACKS THROUGH OLD
                                           SQUARE *)
    ADDATK(RMTO);                       (* ADD ATTACKS FROM NEW SQUARE *)
  END;
END;  (* MOVEIT *)


PROCEDURE RTRKIT                        (* UNPROCESS ORDINARY MOVE *)
  (A:RM);                               (* THE MOVE TO RETRACT *)

BEGIN
  WITH A DO
  BEGIN
    ADDLOC(RMFR,NBORD[RMTO]);           (* PUT PIECE ON ORIGINAL
                                           SQUARE *)
    CUTATK(RMFR);                       (* CUT ATTACKS THROUGH ORIGINAL
                                           SQUARE *)
    DELATK(RMTO);                       (* DELETE ATTACKS FROM
                                           DESTINATION SQUARE *)
    PRPATK(RMTO);                       (* PROPAGATE ATTACKS THROUGH
                                           DESTINATION SQUARE *)
    ADDATK(RMFR);                       (* ADD ATTACKS FROM ORIGINAL
                                           SQUARE *)
  END;
END;  (* RTRKIT *)
```

Listing 1, continued:

```
PROCEDURE PAWNIT                        (* UNPROMOTE A PAWN *)
  (A:RM);                               (* PROMOTION MOVE *)

BEGIN
  WITH A DO
  BEGIN
    MBMORP(NBORD[RMTO]);                (* UPDATE SCORE *)
    NBORD[RMTO] := XTUMP[EP,XTPM[NBORD[RMTO]]];
  END;
END;  (* PAWNIT *)


PROCEDURE PROACA                        (* PROCESS CASTLE STATUS
                                           CHANGES *)
  (A:TS);                               (* SQUARE *)

VAR
  INRS : RS;                            (* SCRATCH *)
  IMRS : RS;                            (* SCRATCH *)

BEGIN
  CLRRS(CSTAT[JNTK],A);                 (* CLEAR THIS SQUARE *)
  ANDRS(INRS,CSTAT[JNTK],XRRS[XTSR[A]]);
                                        (* CASTLE BITS FOR THIS SIDE *)
  IF NOT INRSTB(INRS,XTRFS[XTSR[A],F5]) THEN
                                        (* IF KING MOVE *)
    ANDRS(CSTAT[JNTK],CSTAT[JNTK],XNRS[XTSR[A]]);
                                        (* CLEAR ALL CASTLE MOVES FOR
                                           SIDE *)
  ANDRS(IMRS,INRS,XRFS[F8]);            (* KING ROOK SQUARE *)
  ANDRS(INRS,INRS,XRFS[F1]);            (* QUEEN ROOK SQUARE *)
  IORRS(INRS,INRS,IMRS);                (* BOTH ROOK SQUARES *)
  IF NULRS(INRS) THEN                   (* IF BOTH ROOKS GONE *)
    ANDRS(CSTAT[JNTK],CSTAT[JNTK],XNRS[XTSR[A]]);
END;  (* PROACA *)


PROCEDURE PROACS                        (* PROCESS MOVES AFFECTING CASTLE
                                           STATUS *)
  (A:RM);                               (* MOVE WITH RMAC *)

BEGIN
  WITH A DO
  BEGIN
    IF INRSTB(CSTAT[JNTK],RMFR) THEN    (* FROM SQUARE *)
      PROACA(RMFR);
    IF INRSTB(CSTAT[JNTK],RMTO) THEN    (* TO SQUARE *)
      PROACA(RMTO);
  END;
END;  (* PROACS *)


PROCEDURE PROMOT                        (* PROCESS PROMOTION *)
  (A:RM);                               (* PROMOTION MOVE *)

BEGIN
  WITH A DO
  BEGIN
    MBPROM(XTGMP[RMPP,JNTM]);           (* UPDATE SCORE *)
    NBORD[RMFR] := XTGMP[RMPP,JNTM];
  END;
END;  (* PROMOT *)


PROCEDURE CREATE;                       (* CREATE GLOBAL DATA BASE *)

VAR
  INRS : RS;                            (* SCRATCH BIT BOARD *)
  INTM : TM;                            (* COLOR INDEX *)
  INTP : TP;                            (* PIECE INDEX *)
  INTQ : TQ;                            (* CASTLE TYPE INDEX *)
  INTS : TS;                            (* SQUARE INDEX *)

BEGIN
  WITH BOARD DO
  BEGIN

    JNTM := AM+1;                       (* INITIALIZE MOVES STACK
                                           POINTER *)
    JNTK := AK;                         (* PLY INDEX *)
    JNTM := RBTM;                       (* SIDE TO MOVE *)

    NODES := 0;                         (* INITIALIZE TOTAL NODES *)

    LINDX[JNTK] := JNTM;                (* MOVES ARRAY LIMIT *)
    SRCHM[JNTK] := HQ;                  (* SEARCH MODE *)

    FOR INTS := AS TO ZS DO
    BEGIN
      NEWRS(ATKFR[INTS]);               (* CLEAR ATTACKS FROM *)
      NEWRS(ATKTO[INTS]);               (* CLEAR ATTACKS TO *)
      NBORD[INTS] := MT;                (* CLEAR LOOKAHEAD BOARD *)
    END;

    NEWRS(ALLOC[JNTK]);                 (* CLEAR ALL PIECE LOCATIONS *)

    FOR INTP := LP TO MT DO
      NEWRS(↑PLOC[INTP]);               (* CLEAR PIECE LOCATIONS *)

    FOR INTM := LITE TO NONE DO
    BEGIN
      NEWRS(TMLOC[INTM]);               (* CLEAR COLOR LOCATIONS *)
      NEWRS(ALATK[INTM]);               (* CLEAR COLOR ATTACKS *)
    END;

    MBTOT := 0;
    MBPWN[LITE] := 0;
    MBPWN[DARK] := 0;
    MBLTE := 0;

    FOR INTS := AS TO ZS DO
      IF RBIS[INTS] <> MT THEN
      BEGIN
        ADDLOC(INTS,RBIS[INTS]);
        MBTPAC(RBIS[INTS]);
      END
      ELSE
        SETRS(TPLOC[MT],INTS);

    MBEVAL;                             (* EVALUATE MATERIAL *)

    CPYRS(INRS,ALLOC[JNTK]);            (* COPY BIT BOARD OF ALL
                                           PIECES *)
    WHILE NXTTS(INRS,INTS) DO
      ADDATK(INTS);                     (* ADD ATTACKS OF ALL PIECES *)

    NEWRS(CSTAT[JNTK]);                 (* INITIALIZE CASTLING SQUARES *)
    FOR INTQ := LS TO DL DO
      IF INTQ IN RBSQ THEN
        IORRS(CSTAT[JNTK],CSTAT[JNTK],XSQS[INTQ]);

    NEWRS(ENPAS[JNTK]);                 (* INITIALIZE ENPASSANT SQUARE *)
    IF RBTS >= 0 THEN
      SETRS(ENPAS[JNTK],RBTS);

    CPYRS(GENPN[JNTK],TPLOC[XTUMP[EP,JNTM]]);
    NOTRS(GENTO[JNTK],TMLOC[JNTM]);
    NOTRS(INRS,GENPN[JNTK]);
    ANDRS(GENFR[JNTK],TMLOC[JNTM],INRS);
  END;
END;  (* CREATE *)

PROCEDURE DNDATE                        (* DOWNDATE DATA BASE TO BACK
                                           OUT A MOVE *)
  (A:RM);                               (* THE MOVE TO RETRACT *)

VAR
  INTS : TS;                            (* SCRATCH *)
  INTR : TR;                            (* ROOK RANK FOR CASTLING *)
  INTF : TF;                            (* ROOK FILE FOR CASTLING *)

  RKFR : TS;                            (* ROOK FROM SQUARE *)
  RKTO : TS;                            (* ROOK TO SQUARE *)

BEGIN
  WITH A DO
  BEGIN
    CASE ORD(RMCA)*4 + ORD(RMAC)*2 + ORD(RMPR) OF
      0:  (* ORDINARY MOVE *)
        RTRKIT(A);
      1:  (* PAWN MOVE AND PROMOTE *)
        BEGIN
          PAWNIT(A);
          RTRKIT(A);
        END;
      2:  (* MISCELLANEOUS ACS *)
        IF RMOO THEN
        BEGIN  (* CASTLE *)
          IF RMQS THEN
            INTF := F1                  (* ROOK ON QUEEN ROOK FILE *)
          ELSE
            INTF := F8;                 (* ROOK ON KING ROOK FILE *)
          INTR := XTSR[RMFR];           (* ROOK FILE *)
          RKFR := XTRFS[INTR,INTF];     (* ROOK FROM SQUARE *)
          RKTO := (RMFR+RMTO) DIV 2;    (* ROOK TO SQUARE *)
          ADDLOC(RKFR,NBORD[RKTO]);     (* REPLACE ROOK *)
          DELATK(RKTO);
          PRPATK(RKTO);
          ADDATK(RKFR);
          RTRKIT(A);                    (* RETRACT KING MOVE *)
        END
        ELSE  (* NOT CASTLE *)
          RTRKIT(A);
      3:  (* NULL MOVE *)
      4:  (* CAPTURE *)
        IF RMEP THEN
        BEGIN  (* CAPTURE ENPASSANT *)
          INTS := XTRFS[XTSR[RMFR],XTSF[RMTO]];
          ADDLOC(INTS,RMCP);
          CUTATK(INTS);
          ADDATK(INTS);
          RTRKIT(A);                    (* RETRACT PAWN MOVE *)
          MBTPAC(NBORD[INTS]);          (* ADD PIECE TO SCORE *)
        END
        ELSE  (* CAPTURE NOT ENPASSANT *)
          GAINIT(A);
      5:  (* CAPTURE AND PROMOTE *)
        BEGIN
          PAWNIT(A);                    (* UNPROMOTE *)
          GAINIT(A);                    (* UNCAPTURE *)
        END;
      6:  (* CAPTURE ACS *)
        GAINIT(A);                      (* UNCAPTURE *)
      7:  (* CAPTURE ROOK ACS, PROMOTE *)
        BEGIN
          PAWNIT(A);
          GAINIT(A);
        END;
    END;
    JNTM := LINDX[JNTK];                (* RESET MOVE GENERATION
                                           POINTER *)
    JNTK := JNTK-1;                     (* BACK UP PLY INDEX *)
    JNTM := OTHER[JNTM];                (* SWITCH SIDE TO MOVE *)
  END;
END;  (* DNDATE *)


FUNCTION UPDATE                         (* UPDATE DATA BASE FOR A MOVE *)
  (VAR A:RM)                            (* THE MOVE *)
  :TB;                                  (* RETURNS TRUE IF MOVE IS
                                           LEGAL *)

VAR
  INRS : RS;                            (* SCRATCH *)
  IMRS : RS;                            (* SCRATCH *)
  INTS : TS;                            (* SCRATCH *)
  INTF : TF;                            (* ROOK FILE FOR CASTLING *)
```

Listing 1, continued:

```
  INTR : TR;                          (* ROOK RANK FOR CASTLING *)
  RKTO : TS;                          (* ROOK DESTINATION SQUARE *)
  RKFR : TS;                          (* ROOK ORIGIN SQUARE *)

BEGIN
  WITH A DO
  BEGIN
    JNTK := JNTK+1;                   (* ADVANCE PLY INDEX *)
    NEWRS(ENPAS[JNTK]);               (* CLEAR ENPASSANT BIT BOARD *)
    CPYRS(CSTAT[JNTK],CSTAT[JNTK-1]); (* INITIALIZE CASTLE STATUS *)
    CPYRS(ALLOC[JNTK],ALLOC[JNTK-1]); (* INITIALIZE ALL LOCATIONS *)
    MBVAL[JNTK] := MBVAL[JNTK-1];     (* INITIALIZE MATERIAL SCORE *)
    LINDX[JNTK] := JNTW;              (* MOVES ARRAY LIMIT *)
    CASE ORD(RMCA)*4 + ORD(RMAC)*2 + ORD(RMPR) OF
      0:  (* ORDINARY MOVE *)
        IF RMEP THEN
        BEGIN                         (* PAWN MOVE 2 SPACES *)
          SFTRS(INRS,XRSS[RMTO],S1);
          SFTRS(IMRS,XRSS[RMTO],S3);
          IORRS(INRS,INRS,IMRS);      (* SQUARES NEXT TO DESTINATION *)
          ANDRS(INRS,INRS,TPLOC[XTUMP[EP,OTHER[JNTM]]]);
                                      (* INTERSECT WITH ENEMY PAWNS *)
          IF NOT NULRS(INRS) THEN
            SETRS(ENPAS[JNTK],(RMTO+RMFR) DIV 2);
                                      (* SET ENPASSANT SQUARE *)
          MOVEIT(A);                  (* MOVE PAWN *)
        END
        ELSE
          MOVEIT(A);                  (* MOVE PIECE *)
      1:  (* MOVE AND PROMOTE *)
        BEGIN
          PROMOT(A);                  (* PROMOTE PAWN *)
          MOVEIT(A);                  (* MOVE PROMOTED PIECE *)
        END;
      2:  (* MISCELLANEOUS ACS *)
        BEGIN
          IF RMOO THEN
          BEGIN  (* CASTLE *)
            IF RMQS THEN
              INTF := F1              (* ROOK ON QUEEN ROOK FILE *)
            ELSE
              INTF := F8;             (* ROOK ON KING ROOK FILE *)
            INTR := XTSR[RMFR];       (* ROOK ON KINGS RANK *)
            RKFR := XTRFS[INTR,INTF]; (* ROOK ORIGIN SQUARE *)
            RKTO := (RMFR+RMTO) DIV 2;(* ROOK DESTINATION SQUARE *)
            ANDRS(CSTAT[JNTK],CSTAT[JNTK],XNRS[INTR]);
                                      (* DISALLOW FURTHER CASTLING
                                         BY THIS SIDE *)
            ADDLOC(RKTO,NBORD[RKFR]); (* PUT ROOK ON NEW SQUARE *)
            ADDATK(RKTO);             (* ADD ITS ATTACKS *)
            DELATK(RKFR);             (* DELETE FROM ORIGINAL SQUARE *)
            MOVEIT(A);                (* MOVE KING *)
          END
          ELSE  (* NOT CASTLE *)
          BEGIN
            PROACS(A);                (* PROCESS CASTLE STATUS MODS *)
            MOVEIT(A);                (* MOVE TO OR FROM KING OR ROOK
                                         SQUARE *)
          END;
        END;
      3: (* NULL MOVE *)
      4:  (* CAPTURE *)
        IF RMEP THEN
        BEGIN  (* CAPTURE ENPASSANT *)
          INTS := XTRFS[XTSR[RMFR],XTSF[RMTO]];
                                      (* CAPTURED PAWN SQUARE *)
          MBCAPT(NBORD[INTS]);        (* UPDATE SCORE *)
          DELATK(INTS);               (* DELETE CAPTURED PAWN
                                         ATTACKS *)
          PRPATK(INTS);               (* PROPAGATE ATTACKS THROUGH
                                         PAWN *)
          MOVEIT(A);                  (* MOVE CAPTURING PAWN *)
        END
        ELSE  (* CAPTURE NOT ENPASSANT *)
          LOSEIT(A);                  (* PROCESS CAPTURE *)
      5:  (* CAPTURE AND PROMOTE *)
        BEGIN
          PROMOT(A);                  (* PROMOTE PAWN *)
          LOSEIT(A);                  (* PROCESS CAPTURE WITH PROMOTED
                                         PIECE *)
        END;
      6:  (* CAPTURE ACS *)
        BEGIN
          PROACS(A);                  (* PROCESS CASTLE STATUS MODS *)
          LOSEIT(A);                  (* PROCESS ROOK CAPTURE *)
        END;
      7:  (* CAPTURE ROOK ACS, PROMOTE *)
        BEGIN
          PROMOT(A);                  (* PROMOTE PAWN *)
          PROACS(A);                  (* CHANGE CASTLE STATUS *)
          LOSEIT(A);                  (* PROCESS ROOK CAPTURE *)
        END;
  END;

  (* INITIALIZE MOVE GENERATION *)

  JNTM := OTHER[JNTM];                (* SWITCH SIDE TO MOVE *)
  CPYRS(GENPN[JNTK],TPLOC[XTUMP[EP,JNTM]]);
  NOTRS(GENTO[JNTK],TMLOC[JNTM]);
  NOTRS(INRS,GENPN[JNTK]);
  ANDRS(GENFR[JNTK],TMLOC[JNTM],INRS);

  (* DETERMINE IF MOVE LEAVES KING IN CHECK, OR MOVES
     KING INTO CHECK *)

  ANDRS(INRS,TPLOC[XTUMP[EK,JNTM]],ALATK[OTHER[JNTM]]);
  RMCH := NOT NULRS(INRS);
  ANDRS(INRS,TPLOC[XTUMP[EK,OTHER[JNTM]]],ALATK[JNTM]);
  RMIL := NOT NULRS(INRS);
  UPDATE := NOT RMIL;
  IF NOT RMIL THEN                    (* COUNT LEGAL MOVES *)
    MVSEL[JNTK-1] := MVSEL[JNTK-1] + 1;

  (* INITIALIZE MOVE SEARCHING *)

  SRCHM[JNTK] := H1;
    NODES := NODES+1;                 (* COUNT NODES SEARCHED *)
  END;
END;  (* UPDATE *)


PROCEDURE GENONE                      (* STACK ONE GENERATED MOVE *)
  (A:TT;                              (* FROM SQUARE *)
   B:TS);                             (* TO SQUARE *)

VAR
  INRS : RS;                          (* SCRATCH *)

BEGIN
  WITH MOVES[JNTW] DO
  BEGIN
    RMFR := A;                        (* FROM SQUARE *)
    RMTO := B;                        (* TO SQUARE *)
    RMCP := NBORD[B];                 (* CAPTURED PIECE *)
    RMCA := (NBORD[B] <> MT);         (* CAPTURE *)
    IORRS(INRS,XRSS[A],XRSS[B]);
    ANDRS(INRS,INRS,CSTAT[JNTK]);
    RMAC := NOT NULRS(INRS);          (* AFFECTS CASTLE STATUS *)
    RMCH := FALSE;                    (* CHECK *)
    RMMT := FALSE;                    (* MATE *)
    RMIL := FALSE;                    (* ILLEGAL *)
    RMSU := FALSE;                    (* SEARCHED *)
    RMPR := FALSE;                    (* PROMOTION *)
    RMOO := FALSE;                    (* CASTLE *)
    RMEP := FALSE;                    (* ENPASSANT *)
  END;
  VALUE[JNTW] := 0;                   (* CLEAR VALUE *)
  IF JNTW < ZW THEN
    JNTW := JNTW+1;                   (* ADVANCE MOVES STACK POINTER *)
END;  (* GENONE *)


PROCEDURE PWNPRO;                     (* GENERATE ALL PROMOTION
                                         MOVES *)

VAR
  INTG : TG;                          (* PROMOTION TYPE *)

BEGIN
  MOVES[JNTW-1].RMPR := TRUE;         (* SET PROMOTION *)
  MOVES[JNTW-1].RMPP := PQ;           (* PROMOTE TO QUEEN FIRST *)
  FOR INTG := PR TO PB DO             (* GENERATE OTHER PROMOTIONS *)
  BEGIN
    MOVES[JNTW] := MOVES[JNTW-1];     (* COPY LAST MOVE *)
    MOVES[JNTW].RMPP := INTG;         (* CHANGE PROMOTE TO PIECE *)
    JNTW := JNTW+1;                   (* ADVANCE MOVE INDEX *)
  END;
END;  (* PWNPRO *)


PROCEDURE GENPWN                      (* GENERATE PAWN MOVES *)
  (A:RS;                              (* PAWNS TO MOVE *)
   B:RS);                             (* VALID DESTINATION SQUARES *)

VAR
  INRS, IMRS : RS;                    (* SCRATCH *)
  INTS : TS;                          (* DESTINATION SQUARE *)

BEGIN
  IF JNTM = LITE THEN
  BEGIN                               (* WHITE PAWNS *)
    SFTRS(INRS,A,S2);                 (* ADVANCE ONE RANK *)
    ANDRS(INRS,TPLOC[MT],INRS);       (* ONLY TO EMPTY SQUARES *)
    CPYRS(IMRS,INRS);                 (* SAVE FOR 2 SQUARE MOVES *)
    ANDRS(INRS,B,INRS);               (* ONLY VALID DESTINATION
                                         SQUARES *)
    WHILE NXTTS(INRS,INTS) DO
    BEGIN
      GENONE(XTLS[XTSL[INTS]-XTED[S2]],INTS);
                                      (* GENERATE SIMPLE PAWN MOVES *)
      IF INTS >= XTRFS[R8,F1] THEN
        PWNPRO;                       (* PROCESS PROMOTION *)
    END;
    ANDRS(INRS,IMRS,XRRS[R3]);        (* TAKE ONLY PAWNS ON THIRD *.
    SFTRS(INRS,INRS,S2);              (* ADVANCE ONE MORE RANK *)
    ANDRS(INRS,INRS,TPLOC[MT]);       (* ONLY TO EMPTY SQUARES *)
    ANDRS(INRS,INRS,B);               (* ONLY VALID DESTINATION
                                         SQUARES *)
    WHILE NXTTS(INRS,INTS) DO
    BEGIN
      GENONE(XTLS[XTSL[INTS]-2*XTED[S2]],INTS);
                                      (* GENERATE DOUBLE PAWN MOVES *)
      MOVES[JNTW-1].RMEP := TRUE;     (* FLAG AS TWO SQUARES *)
    END;

    SFTRS(INRS,A,B1);                 (* TRY CAPTURES TO THE LEFT *)
    IORRS(IMRS,TMLOC[OTHER[JNTM]],ENPAS[JNTK]);
                                      (* OPPONENT PIECES + EP SQUARE *)
    ANDRS(IMRS,IMRS,B);               (* VALID DESTINATION SQUARES *)
    ANDRS(INRS,INRS,IMRS);            (* CAPTURE MOVES TO LEFT *)
    WHILE NXTTS(INRS,INTS) DO
    BEGIN
      GENONE(XTLS[XTSL[INTS]-XTED[B1]],INTS);
                                      (* GENERATE CAPTURE MOVE *)
      MOVES[JNTW-1].RMCA := TRUE;     (* FLAG CAPTURE *)
      MOVES[JNTW-1].RMEP := INRSTB(ENPAS[JNTK],INTS);
                                      (* FLAG ENPASSANT CAPTURE *)
      IF MOVES[JNTW-1].RMEP THEN
        MOVES[JNTW-1].RMCP := DP;     (* SET CAPTURED PIECE TYPE *)
      IF INTS >= XTRFS[R8,F1] THEN
        PWNPRO;                       (* PROCESS PROMOTION *)
    END;

    SFTRS(INRS,A,B2);                 (* TRY CAPTURES TO THE RIGHT *)
    IORRS(IMRS,TMLOC[OTHER[JNTM]],ENPAS[JNTK]);
                                      (* OPPONENT PIECES + EP SQUARE *)
    ANDRS(IMRS,IMRS,B);               (* VALID DESTINATION SQUARES *)
    ANDRS(INRS,INRS,IMRS);            (* CAPTURE MOVES TO LEFT *)
    WHILE NXTTS(INRS,INTS) DO
    BEGIN
      GENONE(XTLS[XTSL[INTS]-XTED[B2]],INTS);
                                      (* GENERATE CAPTURE MOVE *)
```

Listing 1, continued:

```
    MOVES[JNTM-1].RMCA := TRUE;          (* FLAG CAPTURE *)
    MOVES[JNTM-1].RMEP := INRSTB(ENPAS[JNTK],INTS);
                                         (* FLAG ENPASSANT CAPTURE *)
    IF MOVES[JNTM-1].RMEP THEN
      MOVES[JNTM-1].RMCP := DP;          (* SET CAPTURED PIECE TYPE *)
    IF INTS >= XTRFS[R8,F1] THEN
      PWNPRO;                            (* PROCESS PROMOTION *)
  END;
END
ELSE
BEGIN                                    (* BLACK PAWNS *)
  SFTRS(INRS,A,S4);                      (* ADVANCE ONE RANK *)
  ANDRS(INRS,TPLOC[MT],INRS);            (* ONLY TO EMPTY SQUARES *)
  CPYRS(IMRS,INRS);                      (* SAVE FOR 2 SQUARE MOVES *)
  ANDRS(INRS,B,INRS);                    (* ONLY VALID DESTINATION
                                            SQUARES *)
  WHILE NXTTS(INRS,INTS) DO
  BEGIN
    GENONE(XTLS[XTSL[INTS]-XTED[S4]],INTS);
                                         (* GENERATE SIMPLE PAWN MOVES *)
    IF INTS <= XTRFS[R1,F8] THEN
      PWNPRO;                            (* PROCESS PROMOTION *)
  END;
  ANDRS(INRS,IMRS,XRRS[R6]);             (* TAKE ONLY PAWNS ON THIRD *)
  SFTRS(INRS,INRS,S4);                   (* ADVANCE ONE MORE RANK *)
  ANDRS(INRS,INRS,TPLOC[MT]);            (* ONLY TO EMPTY SQUARES *)
  ANDRS(INRS,INRS,B);                    (* ONLY VALID DESTINATION
                                            SQUARES *)
  WHILE NXTTS(INRS,INTS) DO
  BEGIN
    GENONE(XTLS[XTSL[INTS]-2*XTED[S4]],INTS);
                                         (* GENERATE DOUBLE PAWN MOVES *)
    MOVES[JNTM-1].RMEP := TRUE;          (* FLAG AS TWO SQUARES *)
  END;

  SFTRS(INRS,A,B3);                      (* TRY CAPTURES TO THE LEFT *)
  IORRS(IMRS,TMLOC[OTHER[JNTM]],ENPAS[JNTK]);
                                         (* OPPONENT PIECES + EP SQUARE *)
  ANDRS(IMRS,IMRS,B);                    (* VALID DESTINATION SQUARES *)
  ANDRS(INRS,INRS,IMRS);                 (* CAPTURE MOVES TO LEFT *)
  WHILE NXTTS(INRS,INTS) DO
  BEGIN
    GENONE(XTLS[XTSL[INTS]-XTED[B3]],INTS);
                                         (* GENERATE PAWN CAPTURE MOVE *)
    MOVES[JNTM-1].RMCA := TRUE;          (* FLAG CAPTURE *)
    MOVES[JNTM-1].RMEP := INRSTB(ENPAS[JNTK],INTS);
                                         (* FLAG ENPASSANT CAPTURE *)
    IF MOVES[JNTM-1].RMEP THEN
      MOVES[JNTM-1].RMCP := LP;          (* SET CAPTURED PIECE TYPE *)
    IF INTS <= XTRFS[R1,F8] THEN
      PWNPRO;                            (* PROCESS PROMOTION *)
  END;

  SFTRS(INRS,A,B4);                      (* TRY CAPTURES TO THE RIGHT *)
  IORRS(IMRS,TMLOC[OTHER[JNTM]],ENPAS[JNTK]);
                                         (* OPPONENT PIECES + EP SQUARE *)
  ANDRS(IMRS,IMRS,B);                    (* VALID DESTINATION SQUARES *)
  ANDRS(INRS,INRS,IMRS);                 (* CAPTURE MOVES TO LEFT *)
  WHILE NXTTS(INRS,INTS) DO
  BEGIN
    GENONE(XTLS[XTSL[INTS]-XTED[B4]],INTS);
                                         (* GENERATE PAWN CAPTURE MOVE *)
    MOVES[JNTM-1].RMCA := TRUE;          (* FLAG CAPTURE *)
    MOVES[JNTM-1].RMEP := INRSTB(ENPAS[JNTK],INTS);
                                         (* FLAG ENPASSANT CAPTURE *)
    IF MOVES[JNTM-1].RMEP THEN
      MOVES[JNTM-1].RMCP := LP;          (* SET CAPTURED PIECE TYPE *)
    IF INTS <= XTRFS[R1,F8] THEN
      PWNPRO;                            (* PROCESS PROMOTION *)
  END;
 END;
END;  (* GENPWN *)



PROCEDURE GENFSL                         (* GENERATE ALL MOVES FROM
                                            A SET OF SQUARES *)
  (A:RS);                                (* ORIGIN SET OF SQUARES *)

VAR
  INRS : RS;                             (* OUTER LOOP BIT BOARD *)
  IMRS : RS;                             (* INNER LOOP BIT BOARD *)
  IPRS : RS;                             (* PAWN ORIGIN BIT BOARD *)
  INTS : TS;                             (* OUTER LOOP SQUARE NUMBER *)
  IMTS : TS;                             (* INNER LOOP SQUARE NUMBER *)

BEGIN
  ANDRS(INRS,A,GENFR[JNTK]);             (* ONLY VALID FROM SQUARES *)
  NOTRS(IMRS,A);
  ANDRS(GENFR[JNTK],GENFR[JNTK],IMRS);   (* REMOVE ORIGIN SQUARES *)
  ANDRS(IPRS,A,GENPN[JNTK]);             (* VALID PAWN FROM SQUARES *)
  ANDRS(GENPN[JNTK],GENPN[JNTK],IMRS);   (* REMOVE PAWNS *)

  WHILE NXTTS(INRS,INTS) DO              (* LOOP THROUGH ORIGINS *)
  BEGIN
    ANDRS(IMRS,ATKFR[INTS],GENTO[JNTK]);
                                         (* GET UNPROCESSED DESTINATION
                                            SQUARES *)
    WHILE NXTTS(IMRS,IMTS) DO            (* LOOP THROUGH DESTINATIONS *)
      GENONE(INTS,IMTS);                 (* GENERATE MOVE *)
  END;
  GENPWN(IPRS,GENTO[JNTK]);              (* GENERATE PAWN MOVES *)
END;  (* GETFSL *)



PROCEDURE GENTSL                         (* GENERATE ALL MOVES TO A
                                            SET OF SQUARES *)
  (A:RS);                                (* TARGET SET OF SQUARES *)

VAR
  INRS : RS;                             (* OUTER LOOP BIT BOARD *)
  IMRS : RS;                             (* INNER LOOP BIT BOARD *)
  IPRS : RS;                             (* PAWN BIT BOARD *)
  INTS : TS;                             (* OUTER LOOP SQUARE NUMBER *)
  IMTS : TS;                             (* INNER LOOP SQUARE NUMBER *)

BEGIN
  ANDRS(INRS,A,GENTO[JNTK]);             (* ONLY VALID TO SQUARES *)
  NOTRS(IMRS,A);
  ANDRS(GENTO[JNTK],GENTO[JNTK],IMRS);   (* REMOVE DESTINATION SQUARES *)
  CPYRS(IPRS,INRS);                      (* SAVE FOR PAWN MOVES *)

  WHILE NXTTS(INRS,INTS) DO              (* LOOP THROUGH DESTINATIONS *)
  BEGIN
    ANDRS(IMRS,ATKTO[INTS],GENFR[JNTK]);
                                         (* GET PIECES OF SIDE TO MOVE *)
    WHILE NXTTS(IMRS,IMTS) DO            (* LOOP THROUGH ORIGINS *)
      GENONE(IMTS,INTS);                 (* GENERATE MOVE *)
  END;
  GENPWN(GENPN[JNTK],IPRS);              (* GENERATE PAWN MOVES *)
END;  (* GENTSL *)



PROCEDURE GENCAP;                        (* GENERATE CAPTURE MOVES *)

VAR
  INRS : RS;                             (* DESTINATION SQUARES *)

BEGIN
  IORRS(INRS,ENPAS[JNTK],TMLOC[OTHER[JNTM]]);
  GENTSL(INRS);                          (* GENERATE MOVES TO
                                            ENEMY SQUARES *)
END;  (* GENCAP *)



PROCEDURE GENCAS;                        (* GENERATE CASTLE MOVES *)

VAR
  INTQ : TQ;                             (* CASTLE TYPE INDEX *)
  INRS : RS;                             (* OCCUPIED SQUARES TEST *)
  IMRS : RS;                             (* ATTACKED SQUARES TEST *)

BEGIN
  FOR INTQ := XTMQ[JNTM] TO SUCC(XTMQ[JNTM]) DO
  IF INRSTB(CSTAT[JNTK],XTQS[INTQ]) THEN
                                         (* IF CASTLING IS LEGAL *)
  BEGIN
    ANDRS(INRS,XRQSO[INTQ],ALLOC[JNTK]);
                                         (* CHECK OCCUPIED SQUARES *)
    ANDRS(IMRS,XRQSA[INTQ],ALATK[OTHER[JNTM]]);
                                         (* CHECK ATTACKED SQUARES *)
    IF NULRS(INRS) AND NULRS(IMRS) THEN
                                         (* IF CASTLING IS LEGAL AND
                                            POSSIBLE *)
    BEGIN
      MOVES[JNTM] := XRQM[INTQ];         (* GENERATE CASTLING MOVE *)
      VALUE[JNTM] := 0;
      JNTM := JNTM+1;
    END;
  END;
END;  (* GENCAS *)



PROCEDURE GENALL;                        (* GENERATE ALL LEGAL MOVES *)

BEGIN
  GENFSL(ALLOC[JNTK]);                   (* GENERATE SIMPLE MOVES *)
  GENCAS;                                (* GENERATE CASTLE MOVES *)
END;  (* GENALL *)



PROCEDURE LSTMOV;                        (* LIST LEGAL PLAYERS MOVES *)

VAR
  INTM : TM;                             (* MOVES INDEX *)

BEGIN
  CREATE;                                (* CREATE DATA BASE *)
  GENALL;                                (* GENERATE ALL MOVES *)
  FOR INTM := AM+1 TO JNTM-1 DO
  BEGIN
    IF UPDATE(MOVES[INTM]) THEN;         (* SET ILLEGAL FLAG *)
    UNDATE(MOVES[INTM]);
  END;
END;  (* LSTMOV *)
    *



PROCEDURE THEMOV                         (* MAKE THE MOVE FOR REAL *)
  (A:RM);                                (* THE MOVE TO MAKE *)

VAR
  INTB : TB;                             (* SCRATCH *)
  INRS : RS;                             (* SCRATCH *)
  INTQ : TQ;                             (* CASTLE TYPE INDEX *)
  INTS : TS;                             (* SCRATCH *)

BEGIN
  LSTMV := A;                            (* SAVE AS PREVIOUS MOVE *)
  INTB := UPDATE(A);                     (* UPDATE THE DATA BASE *)
  WITH BOARD DO                          (* AND COPY ALL THE RELEVANT DATA
                                            BACK DOWN *)
  BEGIN
    RBTM := JNTM;                        (* SIDE TO MOVE *)
    CPYRS(INRS,ENPAS[JNTK]);
    IF NXTTS(INRS,INTS) THEN             (* FIND ENPASSANT SQUARE *)
      RBTS := INTS
    ELSE
      RBTS := AT;
    IF JNTM = DARK THEN
      RBTI := RBTI+1;                    (* ADVANCE MOVE NUMBER *)
    FOR INTQ := LS TO DL DO
      IF INRSTB(CSTAT[JNTK],XTQS[INTQ]) THEN
        RBSQ := RBSQ+[INTQ]              (* CASTLE LEGAL *)
      ELSE
        RBSQ := RBSQ-[INTQ];             (* CASTLE NOT LEGAL *)
    FOR INTS := AS TO ZS DO
      RBIS[INTS] := NBORD[INTS];         (* COPY POSITION *)
  END;
END;  (* THEMOV *)
```

Text continued from page 171

makes use of the routines which were just described (eg: ADDATK, CUTATK, ADDLOC, CLSTAT, PRPATK, DELATK, MOVEIT, LOSEIT). The move is placed on the move stack by GENONE. Special routines exist for generating moves which involve the promotion of a pawn (PWNPRO) and for generating the standard pawn moves (GENPWN). When a move is tried and produces an a-β cutoff, the program backs down the look-ahead tree and begins to explore moves at a different node. Several procedures are employed to downdate the data base. These include the main routines RTRKIT and DNDATE, which are essentially the complement of MOVEIT and UPDATE. Two other procedures are also needed, one to unpromote a pawn (PAWNIT) and one to resurrect a captured piece (GAINIT). This set of routines permits the program to move about the look-ahead tree and incrementally update or downdate the data base.

The executive routines which are responsible for move generation are GENFSL, which generates all legal moves from a set of squares, and GENTSL, which generates all legal moves to a set of squares. The rationale for having two routines is that we wish to generate the moves in stages. For example, captures should be searched first at each node (ie: the capture heuristic). To do this, we identify the square locations of the opponent's pieces, and then call GENTSL to generate all capturing moves. These moves are searched before any other moves are generated. If one of these produces a cut-off the rest of the moves need not be generated at all. A third executive routine (GENCAS) generates all castling moves. These moves are generated after the captures if castling is still legal.

A fourth executive routine for move generation is GENALL. This procedure generates all legal moves and is used by the program to check the legality of the opponent's move. It is called by LSTMOV which makes a list of all the legal moves and each of these are compared with the opponent's move by YRMOVE (presented later). If the opponent's move is not on the list, the machine prints "illegal move." If the opponent's move is compatible with more than one of the moves on the list (eg: P-R3 could be either P-QR3 or P-KR3), the machine prints the message, "ambiguous move." When the machine has completed its own move selection or has determined that the opponent's move is legal and not ambiguous, the move is actually made by THEMOV.■