How to read, make and store chess moves

A. G. Bell* and N. Jacobi†

A vast wealth of knowledge lies untapped in the chess libraries of the world. This paper discusses some simple techniques which could allow computers to access, verify and share this information.

(Received October 1977)

1. Introduction

Chess games have been recorded for many centuries, but the records are not easily accessed or shared by computers. One reason is that the notations used are 'natural', i.e. they were evolved by humans for humans and consequently are untidy, complex, redundant and prone to human error.

This paper describes how a program can read English and algebraic game records by the use of 'skeleton keys'. In order to exemplify the technique (and also reduce the description of trivia to a minimum) a working program is given in the Appendix; the program is in PASCAL (Jensen and Wirth, 1974), a language similar to ALGOL, and should be used more as a footnote to the paper than as an example of efficiency. The paper also proposes an international machine notation which could, in the future, allow computers to share more easily the processed game records.

2. English notation

Essentially this is a descriptive shorthand for

MY MAN (M1) IN SQUARE (S1) TAKES OPPONENTS MAN (M2) in SQUARE (S2) or S1(M1) → S2(M2) where M:: = (empty), P (pawn), N (knight), B (bishop), R (rook), Q (queen), K (king) and S is a (two letter, one digit) square description as shown in **Fig. 1**(*a*).

The common opening move P-K4 is, therefore, a skeleton form of

KK2(P) – KK4() .

In order to identify this move the program builds up the skeleton in the character variables P1, P2, P3, P4, P5, P6, P7, P8 thus

S1			(M1)→	S2			(M2)
P1	P2	P3	P4	P5	P6	P7	P8
·	·	·	P	K	K	4	·

It then compares all the non '·' characters in this skeleton with all the possible pawn moves and finds that only

S1			(M1)→	S2			(M2)
K	K	2	P	K	K	4	0

gives full agreement and the corresponding white move is made, i.e. square 13 to square 29 in the program's board notation. (See Fig. 1(*b*)).

There are four special cases which the skeleton cannot handle. The first is the *en passant* capture (P*PE)—in this case the program shifts the opponent's pawn back one square and then lets nature take its course (see the Appendix for exact details of this case and the two following).

The second special case is king or queen side castling (0-0 or 0-0-0). These moves are flagged by making the piece to move have an apparent value of 9. On returning from SKELETON the program checks for this flag and, if true, makes the correct rook move and primes the king move. N.B. there is no need to match the skeleton but, as the program cannot find a piece of integer value 9 on the board, nature again takes its course.

The third special case is pawn promotion, e.g. P-K8Q. In this case, the program remembers the promotion character 'Q' and, on completion of the move, replaces the pawn with the correct piece.

The fourth special case is when the uniqueness of a move depends on whether or not it is a check/checkmate move or on the fact that similar moves are illegal. The program does not handle such problems. To illustrate the possible complications, consider **Fig. 2.** Black has just played P/2-Q4 and the program, faced with this position and the input P*P, would find an eight-way ambiguity (an octiguity?). Note that a human would also find P*P to be an octiguity but his eight possibilities are not the same as the program's because the human includes the two *en passant* captures and excludes the two illegal captures whereas the program does the opposite.

Fig. 2 also gives a minimal unambiguous notation for each of the eight possible moves plus the skeletons the program would produce plus the correct move in full. Now an experienced human can identify and isolate each of the moves correctly because

(*a*) he knows what *en passant* really means

(*b*) he can use the presence or absence of check (+) or checkmate (+ +), and

(*c*) he knows that the pawn in KN2 cannot make captures.

But the simpleminded program has none of this knowledge and so it still cannot uniquely identify any of the moves from their skeletons. To illustrate an even greater complexity the reader is invited to identify the move P*R+ + ; a move which requires even greater sophistication to identify correctly.

At this point the reader might be puzzled as to why we take such pains to reveal the shortcomings of the program. The important point to appreciate is that the program has no real understanding of what it is doing—it can, for example, quite happily obey the move Q*K. It is therefore necessary to appreciate how successfully this simple program can perform.

As test data for the program 1000 moves were copied out of the Ruy Lopez section of the *Encyclopaedia of Chess Openings*, Vol. C ECO 90-99. These moves are recorded in algebraic notation (see next section) and, when processed, the program was able to isolate and make 973 moves correctly. Of the remaining 27 moves, 17 were genuinely ambiguous in the original text, six were erroneous and only four were ambiguous

*Division of Computing Research, CSIRO Canberra, Australia; now at Computing Services, University of Sheffield, The Hicks Building, Sheffield S10 2TN.
†Department of Economics, University of Newcastle, Newcastle, N.S.W., Australia

QR8	QN8	QB8	QQ8	KK8	KB8	KN8	KR8
QR7	QN7	QB7	QQ7	KK7	KB7	KN7	KR7
QR6	QN6	QB6	QQ6	KK6	KB6	KN6	KR6
QR5	QN5	QB5	QQ5	KK5	KB5	KN5	KR5
QR4	QN4	QB4	QQ4	KK4	KB4	KN4	KR4
QR3	QN3	QB3	QQ3	KK3	KB3	KN3	KR3
QR2	QN2	QB2	QQ2	KK2	KB2	KN2	KR2
QR1	QN1	QB1	QQ1	KK1	KB1	KN1	KR1

(*a*) **English notation for White. The numbers are reversed for Black.**

57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

(*b*) **Program notation**

a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h6
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	c2	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1

(*c*) **Algebraic notation**

70	71	72	73	74	75	76	77
60	61	62	63	64	65	66	67
50	51	52	53	54	55	56	57
40	41	42	43	44	45	46	47
30	31	32	33	34	35	36	37
20	21	22	23	24	25	26	27
10	11	12	13	14	15	16	17
00	01	02	03	04	05	06	07

(*d*) **International notation**

Fig. 1 The various board notations

in the program's terms, i.e. could have been correctly resolved by analysis of the check and illegality information. It therefore seems ridiculous to double (at least) the size of the program to handle 4/1000 = 0·4% of the data correctly, particularly when the majority of problems were produced by incorrect data in the original text or caused by faulty transcription.

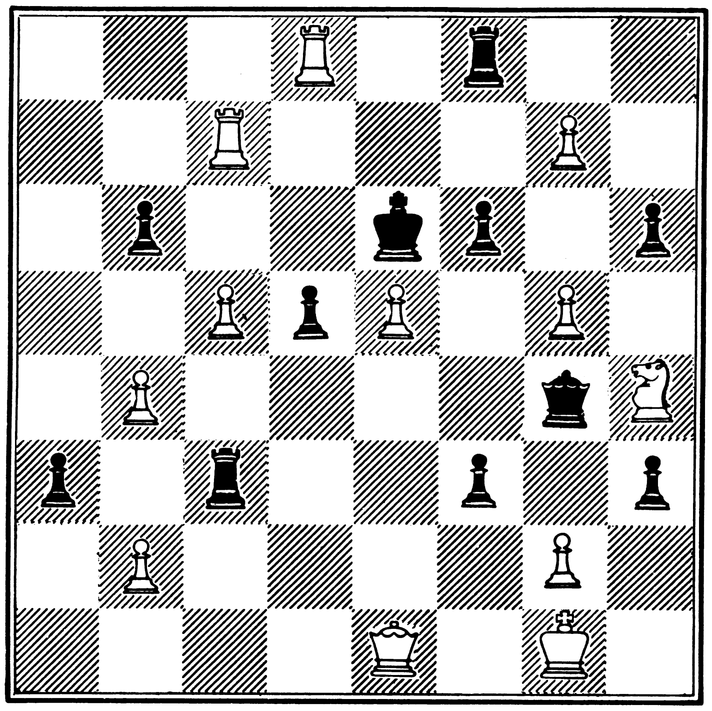
The interesting feature of the 17 genuine ambiguities in the sample is that they are almost all caused by confusion of rooks, for example, in Fig. 2, if white's move is R-Q7. Humans can often resolve these ambiguities by looking at the moves that follow, e.g. if black's reply to R-Q7 is R*R, then we know that it must have been the rook in QB7 that had just moved.

The program also (very crudely) allows what follows to resolve any current ambiguity by simply flagging (it outputs '?') and then making one of the possible moves. Should it eventually find an impossible move it halts and a human scan of the output, with particular attention given to the flagged ambiguities, can usually resolve the problem—an example of ambiguity is given in **Fig. 4.**

3. Algebraic notation

Essentially this is a shorthand for MY MAN (M1) IN SQUARE (S1) GOES TO SQUARE (S2) or (M1)S1 $\rightarrow$ S2, where M is the same as in English notation and S is a (one lower case letter, one digit) square description as shown in Fig. 1(*c*).

Note that in this notation (*a*) pawn moves are simply the square a pawn can move to, e.g. the opening move P-K4 is e4 not Pe4 and (*b*) captures are not distinguished from ordinary moves unless they are pawn captures in which case the file of the pawn is given as identification (Comparison of the English and algebraic examples in the Appendix should make these



	Input	Skeleton	Move
English			
1.	BP*P	. B .(P). . .(P)	QB5(P)QN6(P)
2.	P*PE	. . .(P)Q Q 6(P)	QB5(P)QQ6(P)
3.	P*PE+	. . .(P)Q Q 6(P)	KK5(P)QQ6(P)
4.	P*P+	. . .(P). . .(P)	KK5(P)KB6(P)
5.	P*BP	. . .(P). B .(P)	KN5(P)KB6(P)
6.	P*KRP	. . .(P)K R .(P)	KN5(P)KR6(P)
7.	P*P++	. . .(P). . .(P)	QB4(P)QQ5(P)
8.	P*P/3	. . .(P). . 3(P)	QN2(P)QR3(P)
Algebraic			
1.	CB6	(P)C .B6	(P)C5B6
2.	CD6E	(P)C .D6	(P)C5D6
3.	ED6E+	(P)E .D6	(P)E5D6
4.	EF6+	(P)E .F6	(P)E5F6
5.	GF6	(P)G .F6	(P)G5F6
6.	GH6	(P)G .H6	(P)G5H6
7.	CD5++	(P)C .D5	(P)C4D5
8.	BA3	(P)B .A3	(P)B2A3

Fig. 2 Position which illustrates some weaknesses of the program

conventions clear).

In order to identify the correct move the program builds up an algebraic skeleton in the character variables P1, P2, P3, P4, P5 thus

(M1) S1 → S2

P1	P2	P3	P4	P5
P	·	·	E	4

It then compares all the non '·' characters in the skeleton with all the possible pawn moves and finds that only ,

(M1) S1 → S2

P	E	2	E	4

gives full agreement, i.e. square 13 to square 29.

Algebraic notation (if written accurately) is much more suitable to the program than English notation; in fact, the program can resolve all the pawn captures in Fig. 2. Unfortunately the notation does depend on using lower case letters for squares and upper case letters for pieces. Because the program cannot distinguish upper and lower case letters it is possible to confuse a bishop capture with a b-file pawn capture, e.g. in Fig. 2, if there was a white bishop on square b4, the program would flag an ambiguity and assume that BA3 is more likely to be Ba3 than ba3. Note that (*a*) the correct move can be forced by entering it as PB2A3 and (*b*) none of the other pieces can cause this problem because their initial letters, in PASCAL, are all greater than 'H'.

```
 1   ...P-    KK4.     14  34   ...P-    QQ4.      63  43
 2   ...P-    KK5.     34  44   ...P-    KB4.      65  45
 3   ...P*    KB6PE    44  55   ...Q-    QQ3.      73  53
 4   ...P*    KK.P     55  64   ...B-    QQ2.      72  63
 5   ...P*    ...BB    64  35   ...Q*    ...B      53  75
 6   ...P-    QN4.     11  31   ...P-    QQ5.      43  33
 7   ...P-    .N5.     31  41   ...P-    .B4.      62  42
 8   ...P*    QB6PE    41  52   ...Q*    ...P      75  15
 9   ...P-    .B4.     12  32   ...P*    QB6PE     33  22
10   ...B-    .B4.     05  32   ...P*    .B.P      61  52
11   ...N*    ...P     01  22   ...N-    .B3.      76  55
12   ...N-    .R3.     06  27   ...N-    .R3.      71  50
13   ...N-    KK2.     22  14   ....-    ....-     74  72
14   ..3N-    .B4.     27  35   .R.R-    .B1.      77  75
15   ....-    ....     04  06   ...Q*    ...K      15  06
```

Fig. 3 Output of English program

```
 1   P..E4        13  29   P..D5          52  36
 2   P..E5        29  37   P..F5          54  38
 3   PE.F6        37  46   Q..D6          60  44
 4   PF.E7        46  53   PB.D7          59  52
 5   PE.F8B       53  62   Q..F8          44  62
 6   P..B4        10  26   P..D4          36  28
 7   P..B5        26  34   P..C5          51  35
 8   PB.C6        34  43   Q..F2          62  14
 9   P..C4        11  27   PD.C3          28  19
10   PB.C4         6  27   PB.C6    ?     52  43
11   N..C3         2  19   N..F6          63  46
12   N..H3         7  24   N..A6          58  41
13   N..E2        19  13   0..00          61  59
14   N.3F4        24  30   RH.F8          64  62
15   P..00         5   7   Q..G1          14   7
```

Fig. 4 Output of Algebraic program

One would imagine that algebraic notation is preferable to English notation but, in terms of error detection and correction, this is not true. The confusion of rooks can still occur and is much more difficult to resolve because capture is not indicated, e.g. in Fig. 2, if white plays RD7 then black's reply RD8 does not resolve the ambiguity as it did in the more redundant English notation of R*R.

Even worse is that human error resulting in impossible moves is much more likely in algebraic records—typically the file is out by one letter and the rank by one digit. To recover from such errors requires an extremely knowledgeable scan of what follows and, although a human can often correct such errors, we have no idea how a program might reproduce this ability.

4. International notation

In order to ease the exchange of processed chess games between computers we propose the following data base standard for chess pieces, board and move notation.

1. The pieces are K = 6, Q = 5, R = 4, B = 3, N = 2, P = 1.
2. The board notation is given in Fig. 1(*d*) and is effectively the program's decimal notation less 1 and written in octal.
3. The move notation derives from the piece and board notation. First note that all possible moves can be defined by

four octal digits with the following provisos:

(*a*) if the piece is a pawn moving diagonally to an empty square then it must be an *en passant* capture

(*b*) if the piece is a king moving further than one square then it must be castling

(*c*) if the piece is a pawn moving into a promotion square then the rank of the final square denotes the promotion, e.g. in Fig. 2 6655 means P*RQ, 6645 means P*RR, 6635 means P*RB and 6626 means P-N8N.

Now, although it is possible to compress all moves into four octal digits, this is not good practice partly because this superpacked form requires a fairly elaborate program to decipher it and partly because most computers operate on 8-bit bytes rather than 6 bits.

We therefore propose that a complete move be defined by 16 bits, thus (F) (S1) (P) (S2), where F(Flag) is 2 bits, S1 (from) is 6 bits, P (promotion) is 2 bits and S2 (to) is 6 bits. The deciphering algorithm is then much simpler (note that ( ) means contents of).

```
(S2) := (S1); (S1) := 0;
if (F) = 1 then (* PROMOTION *)
(S2) := (S2) + (P) + 1;
if (F) = 2 then (* EN PASSANT *)
if S2 > S1 then (S2 − 8) := 0 else (S2 + 8) := 0;
if (F) = 3 then (* CASTLING *)
if S2 > S1 then
begin (S2 − 1) := (S2 + 1); (S2 + 1) := 0 end else
begin (S2 + 1) := (S2 − 2); (S2 − 2) := 0 end
```

5. Program description

The program has the structure
MAIN PROGRAM (LISTMOVES(SKELETON,COMPARE,READMOVE,NORKM,RORBM,WORBPM)).

In order to understand how NORKM (knight or king move), RORBM (rook or bishop move) and WORBPM (white or black pawn move) work the reader must consult Algorithm 50 (Bell, 1970).

The tables driving this program are the same as those for Algorithm 50 and the first operation, on entry, is to read them into the arrays KNT (knight), KNG (king), ROOK (rook), BSHP (bishop), WP (white pawn) and BP (black pawn).

The program then clears the board and reads in the position. For the opening position the data is

```
4  1 2  2 3  3 5  4 6  5 3  6 2  7 4  8
1  9 1 10 1 11 1 12 1 13 1 14 1 15 1 16
1 49 1 50 1 51 1 52 1 53 1 54 1 55 1 56
4 57 2 58 3 59 5 60 6 61 3 62 2 63 4 64
```

The program then reads the board notation for each of the 64 squares into the arrays D1, D2, D3, D4. English notation is:

```
QR18QN18QB18QQ18KK18KB18KN18KR18
QR27QN27QB27QQ27KK27KB27KN27KR27
QR36QN36QB36QQ36KK36KB36KN36KR36
QR45QN45QB45QQ45KK45KB45KN45KR45
QR54QN54QB54QQ54KK54KB54KN54KR54
QR63QN63QB63QQ63KK63KB63KN63KR63
QR72QN72QB72QQ72KK72KB72KN72KR72
QR81KN81KB81QQ81KK81KB81KN81KR81
```

The program sets the *en passant* aid (BACK: = 8), the line counter (LINE: = 1) and then begins to process the white and black moves (LISTMOVES (WMEN,BMEN,D3); LISTMOVES (BMEN,WMEN,D4)).

In order to make the program accept algebraic input the procedures SKELETON, COMPARE and READMOVE have to be replaced in the program and the algebraic board notation replaces the English in the data thus:

```
A1 . . B1 . . C1 . . D1 . . E1 . . F1 . . G1 . . H1 . .
A2 . . B2 . . C2 . . D2 . . E2 . . F2 . . G2 . . H2 . .
                  .           .           .
A8 . . B8 . . C8 . . D8 . . E8 . . F8 . . G8 . . H8 . .
```

In order to demonstrate the program the following game was entered in English and algebraic.

	English		*Algebraic*	
1.	P-K4	P-Q4	E4	D5
2.	P-K5	P-KB4	E5	F5
3.	P*PE	Q-Q3	EF6E	QD6
4.	P*KP	B-Q2	FE7	BD7
5.	P*BB	Q*B	EF8B	QF8
6.	P-QN4	P-Q5	B4	D4
7.	P-N5	P-B4	B5	C5
8.	P*PE	Q*P	BC6E	QF2
9.	P-B4	P*PE	C4	DC3E
10.	B-B4	P*BP	BC4	BC6
11.	N*P	N-B3	NC3	NF6
12.	N-R3	N-R3	NH3	NA6
13.	N-K2	0-0-0	NE2	000
14.	N/3-B4	R/R-B1	N3F4	RHF8
15.	0-0	Q*K	00	QG1

The outputs of both versions of the program are given in Figs. 3 and 4. Note that the English output gives the moves in fully compressed International whereas the algebraic output is given in the program's decimal board notation.

One final comment. English notation does not have any fixed standard for its symbols and the characters ( ) , . ? ! and = are often used. When preparing data for the program given here these symbols, together with check (+) and checkmate (+ +), should be omitted whilst any *en passant* indication (usually e.p.) should be represented by just the letter 'E'. Even better would be to modify the procedure READ so that it ignored the characters) , . ? ! = and +, treated (as / and, on reading the letter 'E', threw away all characters until a space was read. The program should then be able to read and verify almost any chess game in English notation with considerably more accuracy than any human.

6. Conclusion

The purpose of this paper is to present the reader with guidelines and illustrations as to how to process, verify and store chess game records. In order to make clear how the skeleton technique works a great deal of the detail has not been discussed. The reader must therefore consult and understand the program given in the Appendix for clarification on a number of trivial problems. He will, if sufficiently perspicacious, find a few errors—for example, the program does not perform correctly for P*BN, i.e. pawn takes a bishop and is promoted to a knight. Our only defence, as already mentioned, is that errors due to the program will be considerably less than errors due to data. It is hoped that, within a decade, programs following the techniques described here will have processed and stored a great deal of the chess games recorded and thus make available to computers a great deal of the knowledge which, up till now, has only been available to literate humans.

Acknowledgements

The authors wish to thank D. Slate, L. Atkin, M. Donsky, J. Birmingham and P. Kent for their comments on the proposed International Notation.

Appendix Pascal program to read English notation

```
PROGRAM ENG(INPUT,OUTPUT);
TYPE KN=ARRAY[1..576]  OF INTEGER;
      BR=ARRAY[0..259]  OF INTEGER;
     WBP=ARRAY[36..227] OF INTEGER;
     MEN=ARRAY[1.. 65]  OF INTEGER;
     D  =ARRAY[1.. 64]  OF CHAR;
VAR I,J,K,L,PA,PB,FROM,SQ,TU,M1,M2,EP,BACK,LINE:  INTEGER;
     CH,P1,P2,P3,P4,P5,P6,P7,P8: CHAR;
     KNT,KNG:     KN;
     BSHP,ROOK:  BR;
     WP,BP:      WBP;
     WMEN,BMEN: MEN;
     D1,D2,D3,D4: D;

PROCEDURE LISTMOVES(VAR MYM,OPP:MEN; DWB:D);
LABEL 1;
VAR C,I,J,K,L,POINT: INTEGER;

PROCEDURE SKELETON;
BEGIN
CH:=' '; P5:='.'; P6:='.'; P7:='.'; P8:='.';
WHILE CH=' ' DO READ(CH);
IF CH='K' THEN BEGIN P5:=CH; P6:=CH; P8:=CH; PB:=6; READ(CH) END;
IF CH='Q' THEN BEGIN P5:=CH; P6:=CH; P8:=CH; PB:=5; READ(CH) END;
IF CH='R' THEN BEGIN           P6:=CH; P8:=CH; PB:=4; READ(CH) END;
IF CH='B' THEN BEGIN           P6:=CH; P8:=CH; PB:=3; READ(CH) END;
IF CH='N' THEN BEGIN           P6:=CH; P8:=CH; PB:=2; READ(CH) END;
IF CH='P' THEN BEGIN                   P8:=CH; PB:=1; READ(CH) END;
IF CH>'0' THEN
IF CH<'9' THEN BEGIN          P7:=CH; P8:='.'; PB:=0; READ(CH) END;
IF CH='0' THEN BEGIN                           PB:=9; READ(CH) END;
IF P5=P6  THEN P5:='.';
IF P6=P8  THEN P6:='.';
IF CH='/' THEN
BEGIN READ(CH);
IF CH='K' THEN BEGIN P5:=CH; P6:=CH; READ(CH) END;
IF CH='Q' THEN BEGIN P5:=CH; P6:=CH; READ(CH) END;
IF CH='R' THEN BEGIN           P6:=CH; READ(CH) END;
IF CH='B' THEN BEGIN           P6:=CH; READ(CH) END;
IF CH='N' THEN BEGIN           P6:=CH; READ(CH) END;
IF CH>'0' THEN
IF CH<'9' THEN BEGIN           P7:=CH; READ(CH) END;
END;
IF CH='E' THEN
BEGIN
OPP[EP+BACK]:=1; OPP[EP]:=0; EP:=EP+BACK;
P5:=D1[EP]; P6:=D2[EP]; P7:=DWB[EP]
END;
WRITE('  ',P5,P6,P7,P8,CH,'  ');
END;      (* SKELETON *)

PROCEDURE COMPARE;
BEGIN
IF (P1='.') OR (P1=D1[FROM])  THEN
IF (P2='.') OR (P2=D2[FROM])  THEN
IF (P3='.') OR (P3=DWB[FROM]) THEN
IF (P5='.') OR (P5=D1[TU])    THEN
IF (P6='.') OR (P6=D2[TU])    THEN
IF (P7='.') OR (P7=DWB[TU])   THEN
IF (PB=OPP[TU]) THEN
BEGIN

IF M1<>0 THEN WRITE('?');
M1:=FROM; M2:=TU;
END
END;    (* COMPARE *)

PROCEDURE READMOVE;
BEGIN
SKELETON; P1:=P5; P2:=P6; P3:=P7; P4:=P8; PA:=PB; SKELETON;
IF PA=9 THEN
BEGIN
IF WMEN[EP]=0 THEN M1:=5 ELSE M1:=61;
IF CH=' ' THEN
BEGIN
M2:=M1+2; MYM[M1+3]:=0; MYM[M1+1]:=4
END ELSE
BEGIN READ(CH);
M2:=M1-2; MYM[M1-4]:=0; MYM[M1-1]:=4
END
END
END;        (* READMOVE *)

PROCEDURE NORKM(VAR NORK:KN);
VAR I:INTEGER;
BEGIN
J:=9*SQ;
FOR I:=J-NORK[J] TO J-1 DO
BEGIN TU:=NORK[I]; IF MYM[TU]=0 THEN COMPARE END;
END; (* NORKM *)

PROCEDURE RORBM(VAR RORB:BR);
LABEL 1;
VAR I:INTEGER;
BEGIN
FOR I:=0 TO 3 DO
BEGIN
L:=RORB[4*SQ+I];
IF L=SQ THEN GOTO 1;
K:=RORB[I]; TU:=SQ;
REPEAT
TU:=TU+K;
IF MYM[TU]=0 THEN COMPARE ELSE GOTO 1;
IF OPP[TU]<>0 THEN GOTO 1;
UNTIL TU=L;
1:END
END;  (* RORBM *)

PROCEDURE WORBPM(VAR PAWN:WBP);
LABEL 1;
VAR I:INTEGER;
BEGIN
J:=4*SQ;
FOR I:=J TO J+1 DO
BEGIN
TU:=PAWN[I];
IF MYM[TU]<>0 THEN GOTO 1;
IF OPP[TU]<>0 THEN GOTO 1 ELSE COMPARE;
END;
1:FOR I:=J+2 TO J+3 DO
BEGIN
TU:=PAWN[I];
IF OPP[TU]<>0 THEN COMPARE;
END
END;  (* WORBPM *)

(* ENTRY TO LISTMOVES *)
BEGIN
M1:=0; M2:=0;
READMOVE;
1: FOR POINT:=1 TO 64 DO
IF MYM[POINT]=PA THEN
BEGIN
FROM:=POINT; SQ:=FROM;
C:=MYM[SQ];
CASE C OF
1:IF WMEN[SQ]=1 THEN WORBPM(WP) ELSE WORBPM(BP);
2:NORKM(KNT);
3:RORBM(BSHP);
4:RORBM(ROOK);
5:BEGIN RORBM(ROOK); RORBM(BSHP) END;
6:NORKM(KNG);
END;
END;

IF P2='B' THEN BEGIN PA:=3; P1:='B'; P2:='.'; GOTO 1 END;
MYM[M2]:=MYM[M1]; MYM[M1]:=0; OPP[M2]:=0; EP:=M2; BACK:=-BACK;
IF CH='Q' THEN BEGIN MYM[M2]:=5; M2:=M2+BACK*2 END;
IF CH='R' THEN BEGIN MYM[M2]:=4; M2:=M2+BACK*3 END;
IF CH='B' THEN BEGIN MYM[M2]:=3; M2:=M2+BACK*4 END;
IF CH='N' THEN BEGIN MYM[M2]:=2; M2:=M2+BACK*5 END;
M1:=M1-1; M2:=M2-1;
I:=M1 DIV 8; J:=M1-I*8; K:=M2 DIV 8; L:=M2-K*8;
WRITE(I:4,J:1,K:4,L:1,'    ');
END;  (* LIST MOVES *)

BEGIN                             (* ENTRY TO PROGRAM  *)
FOR I:= 1 TO 576 DO READ(KNT[I]);
FOR I:= 1 TO 576 DO READ(KNG[I]);
FOR I:= 0 TO 259 DO READ(ROOK[I]);
FOR I:= 0 TO 259 DO READ(BSHP[I]);
FOR I:=36 TO 227 DO READ(WP[I]);
FOR I:=36 TO 227 DO READ(BP[I]);
FOR I:=1 TO 65 DO BEGIN WMEN[I]:=0; BMEN[I]:=0 END;
FOR I:=1 TO 16 DO BEGIN READ(J); READ(K); WMEN[K]:=J; END;
FOR I:=1 TO 16 DO BEGIN READ(J); READ(K); BMEN[K]:=J; END;
READLN;
FOR I:=0 TO 7 DO
BEGIN
FOR J:=1 TO 8 DO
BEGIN
READ(D1[I*8+J]); READ(D2[I*8+J]); READ(D3[I*8+J]); READ(D4[I*8+J])
END;
READLN;
END;

BACK:=8; LINE:=1;
REPEAT
WRITE(LINE:3,'    '); LINE:=LINE+1;
LISTMOVES(WMEN,BMEN,D3);
LISTMOVES(BMEN,WMEN,D4);
WRITELN; READLN
UNTIL EOF;
END.
```

Algebraic replacement

```
PROCEDURE SKELETON;
BEGIN
P1:='P'; P2:='.'; P3:='.'; P4:='.'; P5:='.';
READ(CH); PB:=0;
REPEAT
PB:=PB+1; P2:=P3; P3:=P4; P4:=P5; P5:=CH; READ(CH)
UNTIL CH=' ';
IF P5='E' THEN
BEGIN
OPP[EP+BACK]:=1; OPP[EP]:=0; EP:=EP+BACK;
PB:=5; P5:=P4; P4:=P3; P3:='.';
END;
IF (P5='Q') OR (P5='R') OR (P5='B') OR (P5='N') THEN
BEGIN
CH:=P5; PB:=5; P5:=P4; P4:=P3; P3:='.'
END;
IF PB=3 THEN
BEGIN
IF P3>'H' THEN P1:=P3 ELSE P2:=P3;
P3:='.'
END;
IF PB=4 THEN
BEGIN
P1:=P2; P2:='.';
IF P3<'I' THEN
BEGIN P2:=P3; P3:='.' END
END;
IF P1='P' THEN PA:=1;
IF P1='N' THEN PA:=2;
IF P1='R' THEN PA:=4;
IF P1='Q' THEN PA:=5;
IF P1='K' THEN PA:=6;
IF P5='0' THEN PA:=9;
WRITE(' ',P1,P2,P3,P4,P5,CH,' ');
END;          (*SKELETON *)

PROCEDURE READMOVE;
BEGIN
SKELETON;
IF PA=9 THEN
BEGIN
IF WMEN[EP]=0 THEN M1:=5 ELSE M1:=61;

IF P1<>'0' THEN
BEGIN
M2:=M1+2; MYM[M1+3]:=0; MYM[M1+1]:=4
END ELSE
BEGIN
M2:=M1-2; MYM[M1-4]:=0; MYM[M1-1]:=4
END
END
END;          (* READMOVE *)

PROCEDURE COMPARE;
BEGIN
IF (P2='.') OR (P2=D1[FROM]) THEN
IF (P3='.') OR (P3=D2[FROM]) THEN
IF P4=D1[TU] THEN
IF P5=D2[TU] THEN
BEGIN
IF M1<>0 THEN WRITE('?');
M1:=FROM; M2:=TU
END
END;          (* COMPARE *)
```

References

BELL, A. G. (1970). How to program a computer to play legal chess, *The Computer Journal*, May 1970, pp. 208-219.

JENSEN, K. and WIRTH, N. (1974). *PASCAL User Manual and Report*, Springer Verlag.