Two MBASIC Programs That Write Application Programs for Accessing a Database

R. M. Smith
DSN Facility Operations Section

A method was desired to relieve the tedium of writing and testing application programs. Two utility programs were developed to produce application programs that perform relational operations on data. No coding is performed by the user.

I. Introduction

The nature of the MBASIC Processor facilitates its use by people who are not primarily programmers and who may never have had any previous programming experience. Many managers, and other people needing to make use of management data, fall into this category of user. However, for many data users, programming is so much less important than data usage that it would be beneficial to reduce or eliminate programming in their data activities. One approach to minimizing programming time is to make use of a generalized application program similar to the one described in a previous article (Ref. 1). A generalized program, once written, allows the user to concentrate upon data usage rather than program writing. However, a generalized program requires that the user spell out an access strategy each time the program is used. If a specific application is required for repeated use, then a specialized application program is most desirable. Two different approaches to the use of specialized application programs were described in Refs. 2 and 3. This article describes the result of some preliminary efforts to design a simple method of producing an MBASIC application program while isolating the user from the task of writing code.

Application programs that extract data from a database may be data dependent (knowledge of data organization is built into the application program, making the program sensitive to changes in data organization) or may possess varying degrees of data independence (immunity of an application program to changes in data organization). The greater the degree of data independence, the less the effect of changes in the database. The programs described in this article are preprocessors that accept relational statements from a user, convert the statements to MBASIC code, and store the statements as an application program for later use.

II. Description of the Program 'WRITER'

'WRITER' produces specialized application programs with a moderate amount of data independence. User input to the program is fully prompted and makes use of relational operations (Refs. 4, 5) to specify database access. The relational operations may be invoked in any order repeatedly or used singly. Figure 1 illustrates a session invoking a restriction and a projection in sequence. Figure 2 illustrates the same process in relational notation. Figure 3 is a simplified flow diagram of the program showing the iterative nature of the main program in accepting user input and choosing a specified subprogram to assemble program statements.

In a typical sequence of events, the user

1. Enters the name of the new program to be written.
2. Enters the type of relational operation.
3. Enters the parameters that describe the selected relational operation.

'WRITER' then assembles a program by the following process:

1. Copies a set of generalized code lines to a temporary program file.
2. Appends code lines (to the temporary program file) that are created by 'WRITER' and are specific to the user's application.
3. Appends standardized subroutines (stored for this purpose) for each relational operation involved.
4. Copies the temporary program to a file named by the user.

Figure 4 is a copy of the program produced by 'WRITER' using the process depicted in Fig. 1. Line 100 and lines 902 through 918 are the generalized code lines mentioned previously. Lines 2000 through 2320 and 3000 through 3310 are standardized subroutines for a projection and restriction, respectively. (Lines 5000 through 5090 update a temporary directory relation that describes data files accessed by the application program.)

III. Description of the Program 'WRITPR'

'WRITPR' produces specialized application programs with no data independence. User input is fully prompted and makes use of relational operations (Refs. 4, 5) to specify database access. Figure 5 illustrates a session invoking a restriction and projection (see relational notation in Fig. 2). Figure 6 is a simplified flow diagram of the program. The user prompting sequence for 'WRITPR' is similar to that of 'WRITER' but 'WRITPR' differs in its approach to writing the application program (compare Figs. 3 and 6). 'WRITPR' assembles a program by writing code lines (on the program named by the user) that are specific to the user's application. Figure 7 is a copy of the program produced by 'WRITPR' using the process depicted in Fig. 5.

IV. Miscellaneous Information

To produce code that is specific to the user's application, both programs use "WRITE ON" statements that incorporate variables and "counters" into a completed statement for the application program. Examples of this process are shown in Figs. 8, 9, and 10 and are taken from the program 'WRITPR'. Figure 8 shows the lines of code that produce lines 110 and 120 of the program presented in Fig. 7. The code lines in Fig. 9 produce line 130 of Fig. 7, and the code lines in Fig. 10 produce line 150 of Fig. 7.

Each of the sessions (illustrated in Figs. 1 and 5) requires approximately 3 to 4 min of terminal time and produces programs that are fully functional, requiring no testing of the MBASIC code. Both provide the user with a uniform, extremely simple process for data access. Figure 11 illustrates the data output produced by the application programs written by 'WRITER' and 'WRITPR'.

Data files accessed by these programs must be in, at least, first normal form. The file used in this article (Fig. 12) is in third normal form (Ref. 4).

References

1. Smith, R. M., "An MBASIC Application Program for Relational Inquiries on a Database," in *The Deep Space Network Progress Report 42-34*, Jet Propulsion Laboratory, Pasadena, Calif., August 15, 1976.
2. Smith, R. M., "A Relational Database Implemented Using MBASIC," in *The Deep Space Network Progress Report 42-30*, Jet Propulsion Laboratory, Pasadena, Calif., December 15, 1975.
3. Maiocco, F. R., and Hume, J. P., "Computerizing Goldstone Facility Maintenance Data for Management Decisions," in *The Deep Space Network Progress Report 42-32*, Jet Propulsion Laboratory, Pasadena, Calif., April 15, 1976.
4. Date, C. J., *An Introduction to Database Systems*, Addison-Wesley, 1975.
5. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of ACM*, Vol. 13, No. 6, June 1970.

```
RUN
ENTER NAME OF NEW PROGRAM: INEW

ENTER RELATIONAL OPERATION (R,P,J) OR "STOP": R
 SOURCE RELATION: RMS◆EQPT.ASSIGNMENT
     DOMAIN NAME: OWNER
LOGICAL OPERATOR: =
    DOMAIN VALUE: 12
      PROJECT TO: ◆FILE

ENTER RELATIONAL OPERATION (R,P,J) OR "STOP": P
     SOURCE RELATION: ◆FILE
 QUANTITY OF DOMAINS: 3
     TARGET RELATION: TERMINAL
        DOMAIN NAMES: CON,OPSTAT,LOCATION

ENTER RELATIONAL OPERATION (R,P,J) OR "STOP": STOP
TEST INEW
```

Fig. 1. Illustration of a session at a terminal using 'WRITER' to create an application program (User responses are to the right of each colon.)

'RMS*EQPT.ASSIGNMENT' | OWNER = 12

π TERMINAL (CON, OPSTAT, LOCATION)

Fig. 2. Relational notation describing the process illustrated in Figs. 1 and 5

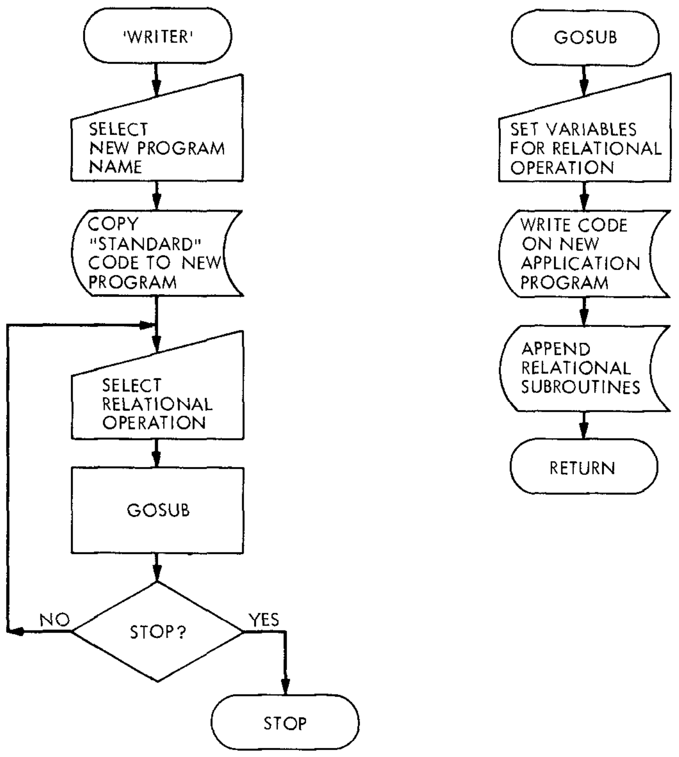

'WRITER'
SELECT NEW PROGRAM NAME
COPY "STANDARD" CODE TO NEW PROGRAM
SELECT RELATIONAL OPERATION
GOSUB
STOP?
NO
YES
STOP
GOSUB
SET VARIABLES FOR RELATIONAL OPERATION
WRITE CODE ON NEW APPLICATION PROGRAM
APPEND RELATIONAL SUBROUTINES
RETURN

Fig. 3. Simplified flow diagram of 'WRITER'

```
COPY 'INEW' TO TERMINAL
100 JOIN=1000,PROJ=2000,REST=3000,NN=44,TARG=5000
110 COPY 'RMS◆EQPT.RREL' TO '◆INEW'
120 DIR$='◆INEW'
130 FL1$='RMS◆EQPT.ASSIGNMENT',DX$='OWNER',LO$='=',DV$='12'
140 RE$='◆FILE'
150 GOSUB REST
160 GOSUB TARG
170 RENAME '◆HOLD1  AS'◆FILE
180 FL1$='◆FILE',QD= 3
190 STRING DP$(QD)/'CON','OPSTAT','LOCATION'/
210 NN=32
220 GOSUB PROJ
902 STRING RCD(1)
904 OPEN '◆HOLD1',INPUT,1
906 AT ENDFILE(1) GO TO 914
908 INPUT FROM 1 USING '(#)':RCD(1)
910 PRINT RCD(1)
912 GO TO 908
914 CLOSE 1
916 REMOVE DIR$
918 STOP CHAR(13)+'END OF RUN'&

2000 OPEN '◆STO1',OUTPUT,3
2010 FLGQ=0
2020 DIM F$(1),D(1),T(1),D$(1)
2030 OPEN DIR$,INPUT,4
2040 AT ENDFILE(4) GO TO 2140
2050 INPUT FROM 4:F$(1),D(1),T(1),D$(1)
2060 STRING DOM(T(1))
2070 IF F$(1)=FL1$ THEN WRITE ON 3:D(1):',':D$(1) ELSE GO TO 2100
2080 FLGQ=FLGQ+1
2090 DOM(FLGQ)=D$(1)
2100 IF F$(1)#FL1$ THEN GO TO 2050
2110 N=D(1)'
2120 IF T(1)=D(1) THEN GO TO 2140
2130 GO TO 2050
2140 CLOSE 3,4
2150 Z=T(1)
2160 DIM DN(QD),D(1),D$(1),K$(Z)
2170 OPEN '◆STO1',INPUT,3
2180 AT ENDFILE(3) GO TO 2240
2190 INPUT FROM 3:D(1),D$(1)
2200 FOR I=1 UNTIL I=QD;1
2210 IF D$(1)=DP$(I) THEN DN(I)=D(1)
2220 NEXT I
2230 GO TO 2190
2240 CLOSE 3
2250 OPEN FL1$,INPUT,3
2260 OPEN '◆HOLD1',OUTPUT,4
2270 AT ENDFILE(3) GO TO 2310
2280 INPUT FROM 3:K$(J) FOR J=1 TO Z
2282 FOR I=1 UNTIL I=QD+1
2284 FOR J=1 UNTIL J=Z+1
2285 IF I<QD THEN CH=NN ELSE CH=13
2286 IF J=DN(I) THEN WRITE ON 4 USING '(#)':K$(J):CHAR(CH)
2288 NEXT J
2290 NEXT I
2300 GO TO 2280
2310 CLOSE 3,4
2320 RETURN&
```

Fig. 4. Data independent application program produced by using 'WRITER'

```
3000 OPEN '*STD1',OUTPUT,3
3002 FLGQ=0
3005 DIM F$(1),D(1),T(1),D$(1)
3010 OPEN DIR$,INPUT,4
3020 AT ENDFILE(4) GO TO 3090
3030 INPUT FROM 4:F$(1),D(1),T(1),D$(1)
3035 STRING DOM(T(1))
3040 IF F$(1)=FL1$ THEN WRITE ON 3:D(1):',':D$(1) ELSE GO TO 3050
3045 FLGQ=FLGQ+1
3046 DOM(FLGQ)=D$(1)
3050 IF F$(1)#FL1$ THEN GO TO 3030
3060 N=D(1)
3070 IF T(1)=D(1) THEN GO TO 3090
3080 GO TO 3030
3090 CLOSE 3,4
3100 DIM D1(1),D2$(1)
3110 STRING A$(N)
3120 OPEN '*STD1',INPUT,3
3130 AT ENDFILE(3) GO TO 3170
3140 INPUT FROM 3:D1(1),D2$(1)
3150 IF D2$(1)=DX$ THEN GO TO 3170
3160 GO TO 3140
3170 CLOSE 3
3180 Q=D1(1)
3190 IF LO$='=' THEN R=0 ELSE GO TO 3210
3200 GO TO 3240
3210 IF LO$='>' THEN R=1 ELSE GO TO 3230
3220 GO TO 3240
3230 IF LO$=CHAR(60) THEN R=-1
3240 OPEN '*HOLD1',OUTPUT,5
3250 OPEN FL1$,INPUT,4
3260 AT ENDFILE(4) GO TO 3300
3270 INPUT FROM 4:A$(I) FOR I=1 TO N
3280 IF COMP(A$(Q),DY$)=R THEN WRITE ON 5 USING '(#)':&
    A$(I):CHAR(NN) FOR I=1 TO N-1\A$(N):CHAR(13)
3290 GO TO 3270
3300 CLOSE 4,5
3310 RETURN&

5000 OPEN '*TRANSF',OUTPUT,1
5010 WRITE ON 1:RE$:',':I:',':FLGQ:',':DOM(I) FOR I=1 TO FLGQ
5020 CLOSE 1
5030 APPEND '*TRANSF' TO DIR$
5040 RETURN&
```

Fig. 4 (contd)

```
LOAD 'WRITPR'
>RUN
ENTER NAME OF NEW PROGRAM: DNEW
SELECT A RELATIONAL OPERATION (REST,PROJ,JOIN) OR ENTER "STOP": REST
     SOURCE RELATION: RMS*EQPT.ASSIGNMENT
          DOMAIN NAME: OWNER
     LOGICAL OPERATOR: =
         DOMAIN VALUE: 12
           PROJECT TO: TERM
PROJECT ALL DOMAINS? N
ENTER QUANTITY OF DOMAINS: 3
ENTER NAMES OF DOMAINS: CON,OPSTA,LOCAT
SELECT A RELATIONAL OPERATION (REST,PROJ,JOIN) OR ENTER "STOP": STOP
TEST 'DNEW'

END OF RUN
>
```

Fig. 5. Illustration of a session at a terminal using 'WRITPR' to create an application program (User responses are to the right of each colon.)

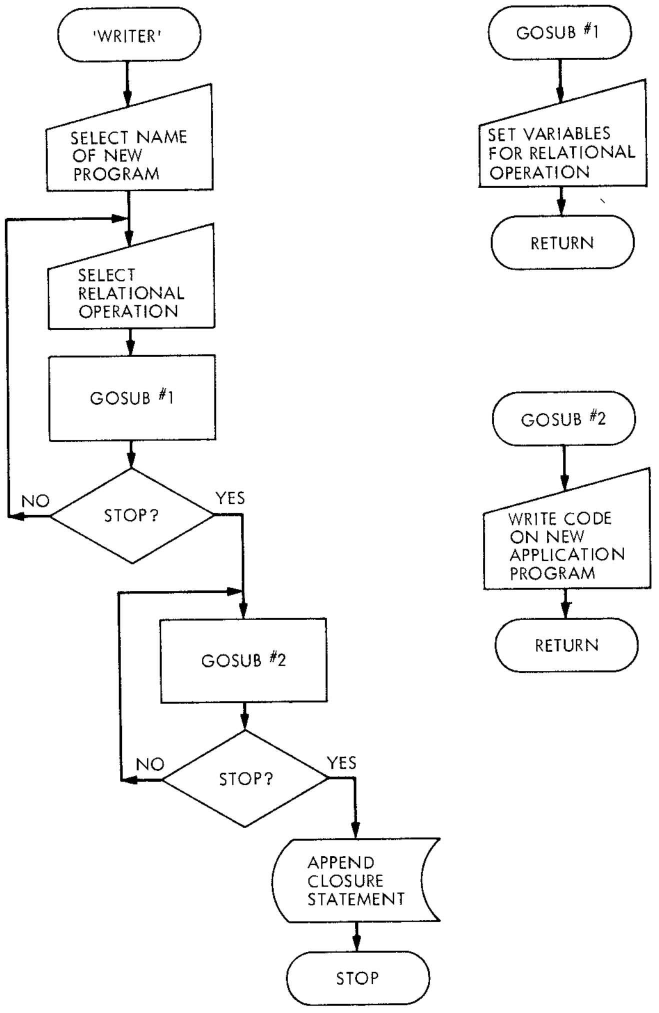

'WRITER'
SELECT NAME OF NEW PROGRAM
SELECT RELATIONAL OPERATION
GOSUB #1
NO
STOP?
YES
GOSUB #2
NO
STOP?
YES
APPEND CLOSURE STATEMENT
STOP
GOSUB #1
SET VARIABLES FOR RELATIONAL OPERATION
RETURN
GOSUB #2
WRITE CODE ON NEW APPLICATION PROGRAM
RETURN

Fig. 6. Simplified flow diagram of 'WRITPR'

```
COPY 'DNEW' TO TERMINAL
100 STRING CON1,OWNER1,LOCAT1,OPSTA1,RECDT1
110 OPEN 'RMS*EQPT.ASSIGNMENT',INPUT, 1
120 AT ENDFILE( 1) GO TO  160
130 INPUT FROM  1:CON1,OWNER1,LOCAT1,OPSTA1,RECDT1
140 IF OWNER1='12' THEN PRINT CON1;OPSTA1;LOCAT1
150 GO TO  130
160 CLOSE I FOR I=1 TO  1
170 STOP CHAR(13)+'END OF RUN'
```

Fig. 7. Data dependent application program produced by using 'WRITPR'

```
3170 LINE=LINE+10
3180 WRITE ON 1:STR(LINE):' OPEN ':'<'>':SR1$(JJ):'<'>':',INPUT,':JJ
3190 LINE=LINE+10
3200 WRITE ON 1:STR(LINE):' AT ENDFILE(':JJ:') GO TO ':
     LINE+30+10*R
>
```

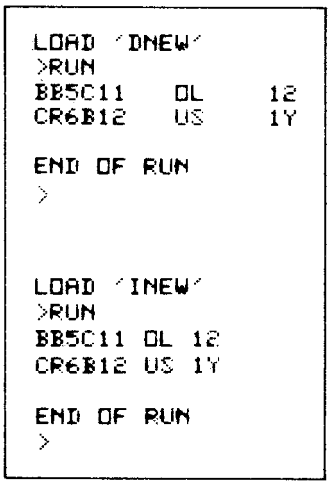
Fig. 8. Sample code lines from 'WRITPR' (lines 110 and 120 of Fig. 7)

```
3210 LINE=LINE+10
3220 WRITE ON 1 USING '(#)':STR(LINE):' INPUT FROM ':JJ:':'
     \DOM(I):STR(JJ):CHAR(44) FOR I=1 TO N-1\DOM(N):
     STR(JJ):CHAR(13)
```

Fig. 9. Sample code lines from 'WRITPR' (line 130 of Fig. 7)

```
3250 LINE=LINE+10
3260 WRITE ON 1:STR(LINE):' GO TO ':LINE-(10+10*R)
```

Fig. 10. Sample code lines from 'WRITPR' (line 150 of Fig. 7)



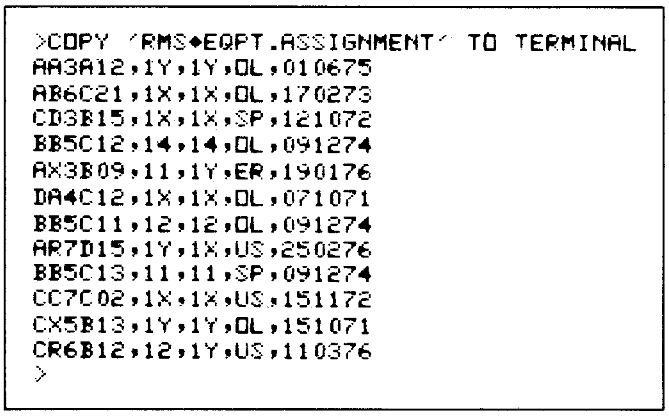
```
LOAD 'DNEW'
>RUN
BB5C11    DL     12
CR6B12    US     1Y

END OF RUN
>


LOAD 'INEW'
>RUN
BB5C11 DL 12
CR6B12 US 1Y

END OF RUN
>
```

Fig. 11. Data printout resulting from running 'WRITPR' and 'WRITER'



```
>COPY 'RMS◆EQPT.ASSIGNMENT' TO TERMINAL
AA3A12,1Y,1Y,DL,010675
AB6C21,1X,1X,DL,170273
CD3B15,1X,1X,SP,121072
BB5C12,14,14,DL,091274
AX3B09,11,1Y,ER,190176
DA4C12,1X,1X,DL,071071
BB5C11,12,12,DL,091274
AR7D15,1Y,1X,US,250276
BB5C13,11,11,SP,091274
CC7C02,1X,1X,US,151172
CX5B13,1Y,1Y,DL,151071
CR6B12,12,1Y,US,110376
>
```

Fig. 12. Structure and content of the relation (data file) used in this article