MSX Article

Agenda EXPERTa

Summary

This article proposes to include an option to save data in diskette for the program “Agenda EXPERTa”, published in the Brazilian Gradiente's Expert MSX manual.

1- Introduction

The program “Agenda EXPERTa” (smart agenda), published in the Expert's manual in the 80's [1], reads names and telephone numbers from the user, in order to illustrate the how to create and use matrices in Basic.

The program “Agenda EXPERTa” is listed below.

Agenda1.bas

```
10 REM Agenda EXPERTa
20 PRINT:PRINT:PRINT"Consult (C) or Insert (I)"; : POKE &HFCAB,255
30 INPUT S$
40 IF S$="C" THEN GOTO 160
50 IF S$="I" THEN GOTO 70
60 GOTO 30
70 PRINT"How many names to insert";
80 INPUT N : IF A>0 THEN ERASE N$
90 A=1 : DIM N$(N,2)
100 FOR F=1 TO N
110 PRINT "Enter name";F;":"
120 INPUT N$(F,1)
130 PRINT "Enter telephone:"
140 INPUT N$(F,2)
150 NEXT F:GOTO10
160 PRINT:PRINT "Name","Telephone"
170 PRINT
180 FOR F=1 TO N
190 PRINT N$(F,1),N$(F,2)
200 NEXT F
300 GOTO 10
```

Menu

Poke used to hold on capital letters

Routine to add members to the agenda

Routine to list the agenda's members

2- Matrices and Files

2.1- Array and Matrix

A variable is designed to store a specific data type in memory. Each programing language has its own set of data type. The Basic classifies the data as numeric or string. The numeric type is also divided into integer, single precision and double precision.

A variable can only store a single data, and the data type must agree with the data type defined for that variable. For example, if in the program agenda we needed to store data for 100 members, we in theory should use 100 variables for each name and 100 for each telephone number. This is not suitable in terms of programing and data management.

An array is a kind of variable that allows to store a collection of data of the same type. In that case, we can store all the names and telephones together and use the same variable name to reference them.

Figure 1 shows the difference between a numeric variable *V* and a numeric array *V(n)*.

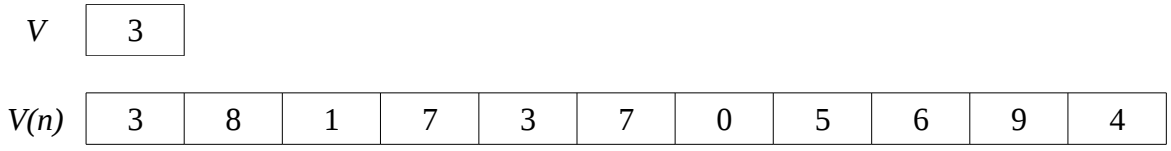

Figure 1. Difference between variable and array.

Suppose that the variables on figure 1 hold the number of TV's sold per day in a store department. While *V* is able to register the TV's sold only one day, the array *V(n)* can register up to 11 days.

For the agenda program, we have a database entity called member and the attributes name and telephone. Once the arrays are unidimensional, it is possible to store only one attribute per array. In our case, we have two attributes.

A matrix is a multidimensional array with two or more dimensions. For the agenda case, we should create a two-dimensional array (table) where each row represents a member and each column represents an attribute (name and telephone).

The Basic instruction DIM, followed by the name and the size inside the parenthesis, creates an array or matrix. The number of arguments passed at the creation time indicates the matrix dimension.

Examples:

- DIM V(10) - creates a numeric array with 11 positions (0 to 10).
- DIM M$(20,4) - creates a string table having 21 rows and 5 cols (0 to 20, 0 to 4).
- DIM C(5,5,5) - creates a numeric cube having the size equal to 6.

Despite of the fact that MSX Basic creates an array or matrix always ranging from 0 to N, the program agenda data ranges from 1 to N.

The use of parenthesis after a variable name indicates that the variable is an array or matrix. The value inside the parenthesis is the array index that user wants to read or write the data. Take a look on figure 2, where *V(n)* is the array data and *n* is the array index (position).

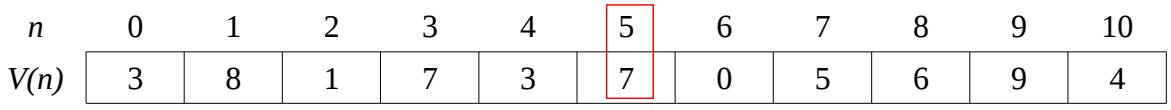

Figure 2. The array and its indexes.

The Basic instruction *PRINT V(5)* prints on screen the array *V* data stored in the position 5, where the value is 7. In another example, the instruction *PRINT M$(1,4)* prints the string matrix *M$* content for row 1 and column 4.

As seen before, the array or matrix size is defined at the creation time and cannot be changed. For that reason, the program asks for the number of members before starting to adding members. Also remember that an array or matrix stores data of the same type. Our agenda table is string type, which means that telephone numbers are stored as string.

Suppose that we run the agenda program and added three members. The generated matrix (*N$*) is represented on figure 3.

	0	1	2
0			
1		CARLA	1234-5678
2		SUELI	1234-9876
3		MARCOS	1243-7766

Figure 3. Table representing the agenda.

The instruction *PRINT N$(1,1)* prints the string “CARLA”.

2.2- Files

The program agenda stores the data in the MSX main memory. Once RAM memory is volatile, whenever the MSX is turned off, the data is lost.

By the time the first MSX's were launched, we could only count on tape recorders. As soon as the disk-drives became more popular, the data could be easily saved in a media that could keep them “alive”, even after turning off the MSX. A single set of data saved is called file.

These are the MSX Basic file formats:

- Binary, where uses the instructions BSAVE/BLOAD.
- Text, where uses the instructions OPEN/CLOSE.

The BSAVE and BLOAD operate on blocks of memory and they are not suitable for accessing variables data. In the other hand, the OPEN and CLOSE allows the direct data manipulation for variables, arrays and matrices. According to that, the last one are more adequate to use with the program agenda.

For many programing languages, the instructions OPEN/CLOSE have the following working flow:

- OPEN the file for reading or writing
- Read or write data in disk (PRINT or INPUT)
- CLOSE the file

Syntax for OPEN:

```
OPEN <file_name> FOR INPUT AS #n    ' Read data from disk
OPEN <file_name> FOR OUTPUT AS #n   ' Write data to disk
```

Where *n* represents the ID of an opened file, ranging from 0 to 15. Despite of that, the MSX is able to handle only 6 disk files at time [2]. The file number *n* is equivalent to the file pointer concept in Pascal or C.

Example:

```
OPEN "DADOS.DAT" FOR OUTPUT AS #1
```

For reading data, the instructions *INPUT #n* or *LINE INPUT #n* are used, followed by a string variable. These instructions read a line from an opened file. Nevertheless, the *INPUT* interrupts the line reading whenever a delimiter character is found (E.g. comma).

Example on how to read a line from a file:

```
INPUT #1, A$
PRINT A$
```

When we try to read a not open file, an error message is shown.

For data writing, the instruction *PRINT #n* is used. Notice that it works very similar to the *INPUT*, where the next line is reached using the semi-colon after the expression. Example:

```
PRINT #1, A$
```

The CLOSE is used to close an opened file. Example:

```
CLOSE #1
```

Despite of the Basic instructions OPEN and CLOSE only manipulate text files format, there are several ways to store data in a file. The Basic course on MarMSX Development page [3] shows in details these formats. This article brings one of these solutions to save data in disk.

3- Adding the disk read/write resource

The original program "Agenda1.bas" is now modified in order to save and load the data from disk. Some other modifications were introduced like a new main menu layout and labels to identify the routines.

The file used to store the agenda's data is the AGENDA.DAT, in a text format.

The next program "Agenda2.bas" adds two new routines: one for saving the table data in disk, and the other for reading the saved data.

Agenda2.bas

```
10 REM Agenda EXPERTa
15 POKE &HFCAB,255
20 CLS:PRINT"Agenda EXPERTa":PRINT:PRINT"Consult (C)":PRINT"Insert
(I)":PRINT"Save (S)":PRINT"Read (R)":PRINT
30 INPUT S$
40 IF S$="C" THEN GOSUB 160
50 IF S$="I" THEN GOSUB 70
55 IF S$="S" THEN GOSUB 300
56 IF S$="R" THEN GOSUB 400
60 GOTO 20
67 '
68 ' Insert
69 '
70 PRINT"How many names to insert";
80 INPUT N : IF A>0 THEN ERASE N$
90 A=1 : DIM N$(N,2)
100 FOR F=1 TO N
110 PRINT "Enter name";F;":"
120 INPUT N$(F,1)
130 PRINT "Enter telephone:"
140 INPUT N$(F,2)
150 NEXT F: RETURN
157 '
158 ' Consult
159 '
160 CLS:PRINT:PRINT "Name","Telephone"
170 PRINT
180 FOR F=1 TO N
190 PRINT N$(F,1),N$(F,2)
200 NEXT F
210 A$ = INPUT$(1) : RETURN
297 '
298 ' Save
299 '
300 OPEN "AGENDA.DAT" FOR OUTPUT AS#1
310 FOR I=1 TO N
320 PRINT#1, N$(I,1)
325 PRINT#1, N$(I,2)
330 NEXT I
340 CLOSE #1
350 RETURN
397 '
398 ' Read
399 '
400 OPEN "AGENDA.DAT" FOR INPUT AS#1
410 IF A>0 THEN ERASE N$: A=1 : N=0
420 INPUT#1, DM$
430 N = N+1
440 IF NOT EOF(1) THEN 420
450 N = N/2 : CLOSE#1
460 DIM N$(N,2)
470 OPEN "AGENDA.DAT" FOR INPUT AS#1
480 FOR I=1 TO N
490 INPUT #1, N$(I,1)
500 INPUT #1, N$(I,2)
510 NEXT I
520 CLOSE#1 : RETURN
```

The writing routine steps is described here:

1. Open the file AGENDA.DAT for writing.
2. Go through the agenda's table data and save each attribute (name, telephone) in a different line.
3. Close the file

The file AGENDA.DAT has this default format:

```
NAME1
TELEPHONE1
NAME2
TELEPHONE2
…
NAMEn
TELEPHONEn
<end_of_file>
```

The reading routine steps:

1. Open the file AGENDA.DAT for reading.
2. Read all the file and count the number of lines.
3. Once each member takes 2 lines, we have *N = N/2*
4. Close the file.
5. Destroy the current table, if exists, and then create a new table using the calculated size.
6. Open the file AGENDA.DAT for reading.
7. Now read each line and copy to the table.
8. Close the file.

3.1- Improving the I/O

The file AGENDA.DAT has no information about the number of members saved in that file. According to that, we are forced to count the number of members before creating a table to keep the names and telephones.

A simple solution for this problem is to add a line at the beginning of the file AGENDA.DAT indicating the number of existing members. Now, by reading the first line, the number of members is known and we do not have to read the file twice to retrieve the data.

The new format for AGENDA.DAT:

```
NUM_REGISTERS
NAME1
TELEPHONE1
NAME2
TELEPHONE2
…
NAMEn
TELEPHONEn
<end_of_file>
```

The program “Agenda3.bas” improves the data reading.

Agenda3.bas

```
10 REM Agenda EXPERTa
15 POKE &HFCAB,255
20 CLS:PRINT"Agenda EXPERTa":PRINT:PRINT"Consult (C)":PRINT"Insert
(I)":PRINT"Save (S)":PRINT"Read (R)":PRINT
30 INPUT S$
40 IF S$="C" THEN GOSUB 160
50 IF S$="I" THEN GOSUB 70
55 IF S$="S" THEN GOSUB 300
56 IF S$="R" THEN GOSUB 400
60 GOTO 20
67 '
68 ' Insert
69 '
70 PRINT"How many names to insert";
80 INPUT N : IF A>0 THEN ERASE N$
90 A=1 : DIM N$(N,2)
100 FOR F=1 TO N
110 PRINT "Enter name";F;":"
120 INPUT N$(F,1)
130 PRINT "Enter telephone:"
140 INPUT N$(F,2)
150 NEXT F: RETURN
157 '
158 ' Consult
159 '
160 CLS:PRINT:PRINT "Name","Telephone"
170 PRINT
180 FOR F=1 TO N
190 PRINT N$(F,1),N$(F,2)
200 NEXT F
210 A$ = INPUT$(1) : RETURN
297 '
298 ' Save
299 '
300 OPEN "AGENDA.DAT" FOR OUTPUT AS#1
310 PRINT#1, N
320 FOR I=1 TO N
330 PRINT#1, N$(I,1)
340 PRINT#1, N$(I,2)
350 NEXT I
360 CLOSE #1
370 RETURN
397 '
398 ' Read
399 '
400 OPEN "AGENDA.DAT" FOR INPUT AS#1
410 IF A>0 THEN ERASE N$: A=1
420 INPUT#1, N
430 DIM N$(N,2)
440 FOR I=1 TO N
450 INPUT #1, N$(I,1)
460 INPUT #1, N$(I,2)
470 NEXT I
480 CLOSE#1 : RETURN
```

An example of a file generated using 3 registers:

```
 3
CARLA
1234-5678
SUELI
1234-9876
MARCOS
1243-7766
```

4- Credits

This article was originally written in portuguese and translated into English by Marcelo Silveira, in October 2016.

Revised in: July 2017, March 2021.
E-mail: flamar98@hotmail.com
Homepage: http://marmsx.msxall.com

Referecences:
[1] - Book: Dominando o Expert, editora Aleph, 5a. Edição, 1987.
[2] - Manual de instruções do Drive Interface DDX.
[3] - Curso de Basic, MarMSX, em http://marmsx.msxall.com.