Microsoft BASIC Variables

Martin Hepperle, January 2026

BASIC-80 and BASIC-86 memory models

The Microsoft BASIC interpreter for CP/M uses a single 64 KB memory area for itself and the user's BASIC code and variables. This layout was designed for 8080 and similar 8-bit systems with a total of 64 KB RAM. For example, Microsoft BASIC-80, Version 5.21 reported `“34872 Bytes free”`.

When the 8088/86 with its segmented memory access opened the memory space to 1 MB, later interpreters, like BASIC-86 and finally, GWBASIC, moved their interpreter code with its associated data into one 64 KB segment and the remaining data, mainly the user code and variables, into a second 64 KB segment. Therefore, Microsoft BASIC, Version 5.28 reported `“62003 Bytes free”`. These segmented versions were usually also ROM-able where the interpreter segment could be executed directly in ROM and only the data segment was using valuable RAM.

This adaptation enabled a cost efficient and easy transition for the BASIC developers and provided more memory for the BASIC application. Nevertheless, only about 128 KB of RAM were used by the improved BASIC interpreter. Finally, starting with Quick-BASIC, Microsoft rewrote their BASIC system to allow larger user programs and data, allowing up to 64 KB of RAM per variable.

MBASIC and GWBASIC

BASIC maintains variables containing the addresses of the start of tables containing simple, array and string variables as well as the current BASIC program. Both families of BASIC are very similar and maintain similar data structures. The global arrangement of the memory area used by a BASIC program is shown in Figure 1 and Table 1.

GWBASIC was often supplied by hardware manufacturers which adapted it to the capabilities of their systems, mostly in the area of graphics output. Thus, many different version of GWBASIC exist. Absolute addresses, as used below, vary from version to version as does the number of keywords and functions. While the source code of an unknown version of GWBASIC (approximately from 1983, maybe close to version 2.0) has been made available by Microsoft, the final version 3.23 and most other versions do not correspond exactly to this source. The internal data structures of the various implementations are similar, but often variables and code have been inserted or deleted, thus moving items in the address space.

Therefore, it may necessary to determine the matching addresses for your individual version. The program `VARTAB.BAS` (Listing 4) can be used to find the addresses of these anchor points.

Microsoft BASIC stores the BASIC program (also called `TEXT`) usually at the bottom of its data area. The 16-bit address of the first line of the BASIC program in memory is stored in `TXTTAB`. The program is stored line by line as a mix of abbreviations for keywords and functions (“tokens”) and ASCII text. The token numbers vary from version to version so that programs stored in binary form are not interchangeable between for example MBASIC and GWBASIC, whereas programs stored in ASCII form usually are (provided they don't use specific keywords or functions available in one version only, like joystick or serial port access).

This program code area is followed by the variables area. At the bottom of this area we find the simple variables, followed by array variables and finally the string working space. The starting address of the table of simple variables is stored in `VARTAB`. The start address of the table of array variables is stored in `ARYTAB`. Creating new simple variables pushes all array variables upwards in memory, invalidating any array addresses obtained previously with the `VARPTR` function.

The "string space" starts above the array table. Its lower end address is stored in `STREND`. This large part of the memory is managed dynamically and used when string operations are performed. Any manipulation of one or more strings stores intermediate strings and results in this area (an exception are string constants, which are stored in the program text area. For example the string "ABC" in a statement A$="ABC" is stored directly as part of the program text, whereas the statement A$="AB"+"C" would store "AB" and "C" in the program text, but the result "ABC" ends up in string space).

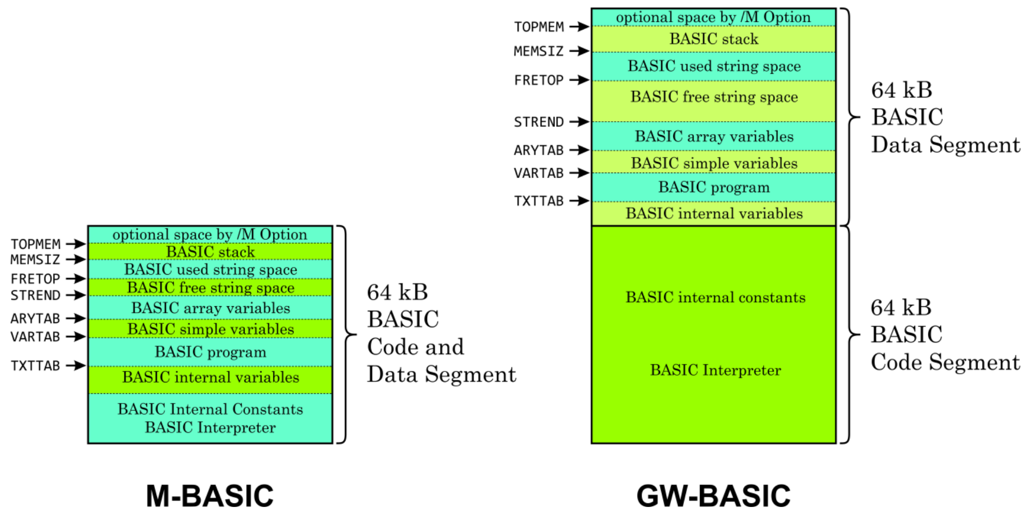

Figure 1: Memory layout of Microsoft BASIC for 8080 and for 8086.

Table 1: Typical arrangement of the data area of a BASIC system (GWASIC 2.32).

Name	Offset		Address	Description
			high addresses	
`TOPMEM`	311	→	65278	top of stack, may be lowered by /M option
			growth ↓	… stack space, usually 256+2 bytes
`MEMSIZ`	1103	→	65020	top of string space
			growth ↓	used string space
`FRETOP`	1140	→	61783	top of free string space
				… free string space
`STREND`	1186	→	14280	bottom of string space
			growth ↑	… table of array variables
`ARYTAB`	1184	→	13060	start of array variables
			growth ↑	… table of simple variables
`VARTAB`	1182	→	12971	start of simple variables
				… BASIC program lines
`TXTTAB`	315	→	4718	first line of BASIC program
				… internal variables used by BASIC interpreter
			low addresses	

String Space and Garbage Collection

Operations on strings occur frequently may result in many temporary string fragments. These also include the generation of strings for output of numeric variables by `PRINT` statements. During the execution of a program the string space fills with strings in use and unreferenced string fragments. The space occupied by unreferenced strings will be reclaimed when the free string space becomes exhausted. A “garbage collector” routine walks through the string space and compares its contents with the list of simple and array string variables. It deletes any unreferenced strings und moves the remaining strings upwards, collecting them in a contiguous area at the upper end of the string space. Thus, gaps are closed and a large, free area is produced towards the lower end of string space. This garbage collection can also be forced by calling the `FRE()` function with any string argument, e.g. an empty string.

Listing 1: Program demonstrating the effect of garbage collection in string space.

```
10 A1=311        : ADDR1=PEEK(A1)+256*PEEK(A1) : PRINT "TOPMEM @";ADDR1
20 PRINT "  -258 =";ADDR1-258
30 A1=1186       : ADDR2=PEEK(A1)+256*PEEK(A1) : PRINT "STREND @";ADDR2
40 N=FRE("")
50 A1=0 : A2=0 : ADDR1=0 : ADDR2=0 : N=0 : I=0
60 REM Create initial strings
70 X$="ABC"+"DE"
80 Y$="VWX"+"YZ"
90 PRINT "Initially:";TAB(25);
100 GOSUB 240
110 REM Shuffle for generating many fragments in string space
120 N=3000
130 FOR I=1 TO N
140  X$=LEFT$(X$+"01234",5)
150 NEXT I
160 PRINT "After";N;"loops:";TAB(25);
170 GOSUB 240
180 REM Force garbage collection
190 N=FRE("")
200 PRINT "After FRE(";CHR$(34);CHR$(34);"):";TAB(25);
210 GOSUB 240
220 STOP
230 REM ----- Show Variables
240 PRINT TAB(24);
250 A1=VARPTR(X$) : ADDR1=PEEK(A1+1)+256*PEEK(A1+2) : PRINT "X$ @";ADDR1;": ";
260 FOR N=ADDR1 TO ADDR1+PEEK(A1)-1
270 PRINT CHR$(PEEK(N));
280 NEXT N
290 PRINT
300 PRINT TAB(24);
310 A2=VARPTR(Y$) : ADDR2=PEEK(A2+1)+256*PEEK(A2+2) : PRINT "Y$ @";ADDR2;": ";
320 FOR N=ADDR2 TO ADDR2+PEEK(A2)-1
330 PRINT CHR$(PEEK(N));
340 NEXT N
350 PRINT
360 RETURN
```

Running the program in Listing 1 with `N=3000` yields the following results:

```
Initially:
   X$ @ 65016 : ABCDE
   Y$ @ 65011 : VWXYZ
After 3000 loops:
   Y$ @ 65011 : VWXYZ     Y$ still in place, about 45 KB of string space has been used
   X$ @ 20011 : ABCDE     X$ is now quite low in string space
After FRE(""):
   X$ @ 65011 : ABCDE
   Y$ @ 65016 : VWXYZ     Y$ has been moved up
```

Initially, `X$` is at the top of the string space, followed below by `Y$`.

Just before entering the loop, the string space looks like this:

```
TOPMEM →    65278                           upper end of stack space
TOPMEM-258→ 65020                           upper end of string space
Addresses   65016-65020: A B C D E          X$ at top of string space
Addresses   65011-65015: V W X Y Z          Y$ next lower entry in string space
...                                         unused
STREND →    11051                           bottom of string space
```

Executing the loop 3000 times exercises the string space, but does not cause a garbage collection because `STREND` is not yet reached. `X$` can be found at a low address in the downwards growing string space, whereas `Y$` is still at its initial address close to the top.

After the loop has been executed, the string space looks like this:

```
TOPMEM →    65278                           upper end of stack space
TOPMEM-258→ 65020                           upper end of string space
...                                         temporarily used, garbage
Addresses   65011-65015: V W X Y Z          Y$ next lower entry in string space
...                                         temporarily used, garbage
Addresses   20011-20015: A B C D E          X$ low in string space
...                                         unused
STREND →    11051                           bottom of string space
```

Calling `FRE("")` forces a garbage collection and string space has not only been compacted, but also rearranged: because `X$` was defined first, accidentally, it ends up at the address at the top of the string space, where `Y$` initially was allocated.

When the loop count is increased to `5000`, the string space is exhausted during the execution of the loop. Now `Y$` has already been moved during the execution of the loop and the `FRE("")` command only moves `X$` up, just below `Y$`.

```
Initially:
   X$ @ 65016 : ABCDE
   Y$ @ 65011 : VWXYZ
After 5000 loops:
   X$ @ 49306 : ABCDE
   Y$ @ 65016 : VWXYZ     Y$ has been moved up due to garbage collection
After FRE(""):
   X$ @ 65011 : ABCDE
   Y$ @ 65016 : VWXYZ
```

At the end of the program, the string space looks like this:

```
TOPMEM →    65278                           upper end of stack space
TOPMEM-258→ 65220                           upper end of string space
Addresses   65016-65020: V W X Y Z          Y$ at top of string space
Addresses   65011-65015: A B C D E          X$ next lower entry in string space
...                                         unused
STREND →    11051                           bottom of string space
```

Variable Names

Standard ANSI BASIC defined variable names to have a length of one character or one character and one digit. Thus, the maximum length of a name was limited to two. Most implementors of BASIC quickly extended these short names so that the naming scheme had to be adapted. This is the reason why Microsoft BASIC stores the names of variables as two characters, followed by an “extra characters” count and the sequence of extra characters. In case of single or dual character names the count is zero and no extra characters are stored.

Simple Variables

Simple variables are all variables which are no arrays. They can be of integer, single, double or string type. The 16-bit start address of the table of simple variables is stored at the address `VARTAB` and the table extends up to the address stored in `ARYTAB`.

Note: If bit 7 of the first character in the name is set, the variable is a user defined `FN` function and the first two bytes in the value contain the address of the token stream in the program text, following the closing parenthesis of the parameter list.

Table 2: Memory structure of a simple variable.

Offset	Bytes	Name	Description
0	1	T	variable type (equals data size: 2,3,4,8 bytes)
			2 = integer, low, high byte
			3 = string: length, address
			4 = single: value, MBF
			8 = double: value, MBF
1	1	N0	first character of name
2	1	N1	second character of name, or zero
3	1	NC	number of additional characters in name (excl. the first two)
4	NC	Nn	only if NC > 0: additional characters, ORed with &H80
4+NC	T	B0 .. Bn	value, T bytes

Array Variables

All variable types (integer, single, double, string) can be used for defining arrays. Array variables are stored above simple variables. Therefore, all array variables are moved upwards each time a new simple variable is allocated. This means, that the address of an array variable is only valid as long as no new simple variables are created.

Multidimensional array elements are stored so that the first index varies first. If `OPTION BASE 1` has been selected, the stored data starts with index one, i.e. no space is wasted. The dimensions of multidimensional arrays are store in right to left order, i.e. `X(1,2,3)` has `DIM1=4`, `Dim2=3` and `DIM3=2` (assuming `OPTION BASE 0`).

The name part up to byte 4+NC-1 is identical to simple variables. The 16-bit start address of the table of array variables is stored at the address `ARYTAB` and the table extends up to the address stored in `STREND`.

Table 3: Memory structure of an array variable.

Offset	Bytes	Name	Description
0	1	T	variable type (equals data size: 2,3,4,8 bytes)
			2 = integer, low, high byte
			3 = string: length, address
			4 = single: value, MBF
			8 = double: value, MBF
1	1	N0	first character of name
2	1	N1	second character of name, or zero
3	1	NC	number of additional characters of name (excl. the first two)

4	NC	Nn	if NC > 0: additional characters, ORed with &H80
4+NC	2	SIZE	total size of the following data in bytes (incl. NDIM, …)
4+NC+2	1	NDIM	number of dimensions of array
4+NC+3	2	DIMn	last dimension of array
4+NC+3+2×n	2×n	DIM2 .. DIM1	only if NDIM>1: n=NDIM-1 lower dimensions of array
4+NC+3+NDIM×2	DIM×T	B0 .. Bn	DIM1×DIM2×… values, T bytes each

Program VARTAB.BAS

This program helps finding some of the relevant addresses for directly accessing BASIC programs and data.

```
GW-BASIC 3.23
(C) Copyright Microsoft 1983,1984,1985,1986,1987,1988
60300 Bytes free
Ok

TXTTAB @ 315 -> 4718  start of BASIC program.
VARTAB @ 1182 -> 7376 start of simple variables
ARYTAB @ 1184 -> 7455 start of array variables
ARYTA2 @ 1425 -> 7455 start of array variables, again
STREND @ 1186 -> 7592 bottom of string space
DATPTR @ 1188 -> 4717 address of next DATA line
DEFTBL @ 1190 start of default types A-Z
USRTAB @ 264 start of USR functions 0-9
DAYSPM @ 294 days per month table
TOPMEM @ 311 -> 65534 top location for stack           ... likely, check
Enter 10 times '<' character? <<<<<<<<<<<<<<
BUF    @ 825 start of input buffer
Linear access to 2D-Array variable:
R(0) = 1
R(1) = 2
R(2) = 3
```

Listing 2: Output of VARTAB when starting GWBASIC without any options.

```
 (C) Copyright Microsoft 1983,1984,1985,1986,1987,1988
24766 Bytes free
Ok

TXTTAB @ 315 -> 4718  start of BASIC program.
VARTAB @ 1182 -> 7510 start of simple variables
ARYTAB @ 1184 -> 7589 start of array variables
ARYTA2 @ 1425 -> 7589 start of array variables, again
STREND @ 1186 -> 7680 bottom of string space
DATPTR @ 1188 -> 4717 address of next DATA line
DEFTBL @ 1190 start of default types A-Z
USRTAB @ 264 start of USR functions 0-9
DAYSPM @ 294 days per month table
TOPMEM @ 311 -> 30000 top location for stack           ... likely, check
Enter '<' character 10 times ? <<<<<<<<<<<<<<
BUF    @ 825 start of input buffer
Linear access to 2D-Array variable R(20,2):
R(0) = 1
R(1) = 2
R(2) = 3
```

Listing 3: Output of VARTAB with GWBASIC option /M=30000 or /M:30000.

```
10 REM
20 REM
30 REM
40 REM ! Do not change the first 3 empty REM lines!
50 REM Microsoft MBASIC, GWBASIC
60 REM Find addresses of VARTAB, ARYTAB, ...
70 REM ! 'I' MUST be the first defined simple variable!
80 REM Allocate all simple variables to avoid movement of R%()
90 I=0 : A=0 : B=0 : C=0 : D=0 : T=0 : X=0 : Y=0 : Z=0 : L$=""
100 DIM R%(19,1) : R%(0,0)=1 : R%(1,0)=2 : R%(2,0)=3
110 REM first look for the sequence of the 3 empty REM lines
120 FOR I=300 TO 400
130 Z=PEEK(I)+256*PEEK(I+1)
140 IF Z=65535! THEN GOTO 190
150 Y=PEEK(Z)+256*PEEK(Z+1)
160 IF Y=65535! THEN GOTO 190
170 X=PEEK(Y)+256*PEEK(Y+1) : REM sequence of lines 6 bytes apart?
180 IF Y=Z+6 AND X=Y+6 THEN PRINT "TXTTAB @";I;"->";Z;" start of BASIC program." : I=500
190 NEXT I
200 REM -----
210 REM VARPTR points to first data byte B0
220 REM 4,'I',0,0,B0,B1,B2,B3                           first simple var
230 A=VARPTR(I)-4
240 REM 2,'R',0,0,SIZ0,SIZ1,NDIM,D1L,D1H,D2L,D2H,B0,B1  first array var
250 B=VARPTR(R%(0,0))-11
260 FOR I=1000 TO 2000
270 X=PEEK(I)+256*PEEK(I+1)
280 IF X=A THEN PRINT "VARTAB @";I;"->";X;"start of simple variables"
290 IF X=B AND T=1 THEN PRINT "ARYTA2 @";I;"->";X;"start of array variables, again"
300 IF X=B AND T=0 THEN PRINT "ARYTAB @";I;"->";X;"start of array variables" : D=I : T=1
310 NEXT I
320 REM ----- string space above R%(): ARYTAB + 11 + 40*2 = 91 bytes
330 C=PEEK(D)+256*PEEK(D+1)+91
340 REM should also be stored at STREND:
350 I=D+2 : X=PEEK(I)+256*PEEK(I+1)
360 IF C=X THEN PRINT "STREND @";I;"->";X;"bottom of string space"
370 I=D+4 : X=PEEK(I)+256*PEEK(I+1)
380 REM DATPTR should be (*TXTTAB)-1
390 IF X=Z-1 THEN PRINT "DATPTR @";I;"->";X;"address of next DATA line"
400 REM ----- all 26 default types are 4
410 FOR I=1000 TO 1200
420 FOR C=I TO I+26-1
430 IF PEEK(C)<>4 THEN GOTO 460
440 NEXT C
450 IF C=I+26 THEN PRINT "DEFTBL @";C-26;"start of default types A-Z" : I=1200
460 NEXT I
470 REM ----- all 10 undefined USR? functions are &HFFFF
480 FOR I=100 TO 400
490 FOR C=I TO I+20-1
500 IF PEEK(C)<>255 THEN GOTO 530
510 NEXT C
520 IF C=I+20 THEN PRINT "USRTAB @";C-20;"start of USR functions 0-9" : I=400
530 NEXT I
540 REM ----- look for days/month table
550 FOR I=100 TO 500
560 IF PEEK(I)<>31 THEN GOTO 620
570 IF PEEK(I+1)<>28 THEN GOTO 620
580 IF PEEK(I+2)<>31 THEN GOTO 620
590 IF PEEK(I+3)<>30 THEN GOTO 620
600 IF PEEK(I+11)<>31 THEN GOTO 620
610 PRINT "DAYSPM @";I;"days per month table" : A=I : I=500
620 NEXT I
630 REM ----- likely 17 bytes behind start of days-per-month table
640 A=A+17
650 PRINT "TOPMEM @";A;"->";PEEK(A)+256*PEEK(A+1);"top location for stack           ...
likely, check"
660 REM ----- look for BUF buffer
670 INPUT "Enter '<' character 10 times ";L$
680 FOR I=100 TO 1000
690 FOR C=I TO I+9
```

```
700 IF PEEK(C)<>60 THEN GOTO 730
710 NEXT C
720 IF C=I+10 THEN PRINT "BUF    @";C-10;"start of input buffer" : I=1000
730 NEXT I
740 REM ----- read array data sequentially
750 PRINT "Linear access to 2D-Array variable R(20,2):"
760 B=B+11 : PRINT "R(0) ="; PEEK(B)+256*PEEK(B+1)
770 B=B+2 : PRINT "R(1) =";PEEK(B)+256*PEEK(B+1)
780 B=B+2 : PRINT "R(2) =";PEEK(B)+256*PEEK(B+1)
790 REM -----
800 END
```

Listing 4: Program VARTAB searches for some known internal variables.

```
10 REM
20 REM Microsoft GWBASIC 3.23 memory dumper
30 REM Addresses valid for GWBASIC.EXE
40 REM date: 05/OCT/1988, size: 80608 bytes
50 REM
60 REM Martin Hepperle, 2020
70 REM
80 REM Make sure that segment address is default
90 DEF SEG
100 DIM ARY(128)
110 DIM CMD$(124),XFD$(11),XFE$(40),XFF$(37)
120 REM get a word at A -> FN becomes a simple variable, bit 7 in 1st character set
130 DEF FNPEEKW(A)=PEEK(A)+256*PEEK(A+1)
140 RESET
150 REM Read single byte tokens
160 FOR I% = 0 TO 123 : READ CMD$(I%) : NEXT I%
170 REM Read escaped tokens
180 FOR I% = 0 TO 10 : READ XFD$(I%) : NEXT I%
190 FOR I% = 0 TO 39 : READ XFE$(I%) : NEXT I%
200 FOR I% = 0 TO 36 : READ XFF$(I%) : NEXT I%
210 PRINT "Start offset for HEX dump";
220 INPUT S
230 E=S+4*64-1
240 L$ = ""
250 N=16
260 PRINT "MEMORY FROM";S;"TO";E;
270 REM PRINT RIGHT$("000"+HEX$(S),4);": ";
280 FOR I = S TO E
290  IF N = 16 THEN PRINT " "; L$ : N = 0 : L$ = "" : PRINT RIGHT$("000"+HEX$(I),4);": ";
300  C% = PEEK(I)
310  IF C% > 31 AND C%<128 THEN C$ = CHR$(C%) ELSE C$ = "."
320  L$ = L$ + C$
330  PRINT MID$("0" + HEX$(C%), 1 - (C% > 15)); " ";
340  N=N+1
350 NEXT I
360 IF N>0 THEN FOR I=1 TO 16-N : PRINT "   "; : NEXT I : PRINT " ";L$
370 PRINT "[Press ENTER to continue]" : LINE INPUT C$
380 PRINT "Variables:"
390 MSG$ = "TOPMEM": OFFS% = 311: GOSUB 640 : REM O.K. -> top of stack
400 MSG$ = "MAXMEM": OFFS% = 1101: GOSUB 640 : REM O.K. -> &HFFFE data seg size
410 MSG$ = "MEMSIZ": OFFS% = 1103: GOSUB 640 : REM O.K. -> &HFDFC max. offset used
420 MSG$ = "FRETOP": OFFS% = 1140: GOSUB 640 : REM O.K. ->
430 REM MSG$ = "DEVTBL": OFFS% = 321: GOSUB 630 : REM ???
440 MSG$ = "STREND": OFFS% = 1186: GOSUB 640 : REM O.K. -> lower end of string vars
450 MSG$ = "ARYTAB": OFFS% = 1184: GOSUB 640 : REM O.K. -> array variables
460 MSG$ = "ARYTA2": OFFS% = 1425: GOSUB 640: REM O.K. same as ARYTAB
470 MSG$ = "VARTAB": OFFS% = 1182: GOSUB 640 : REM O.K. -> simple variables
480 MSG$ = "TXTTAB": OFFS% = 315: GOSUB 640 : REM O.K. -> program TEXT
490 MSG$ = "DATPTR": OFFS% = 1188: GOSUB 640 : REM O.K. -> initial DATA: 0 byte before TEXT
500 MSG$ = "AUTINC": OFFS% = 1159: GOSUB 640 : REM O.K. AUTO line # increment
510 MSG$ = "USRTAB": OFFS% = 264: GOSUB 640 : REM O.K. -> USR0 to USR9 functions
520 MSG$ = "DEFTBL": OFFS% = 1190: GOSUB 640 : REM O.K. -> 26 default types, 4,4 ...
530 MSG$ = "SAVSEG": OFFS% = 1174: GOSUB 640 : REM O.K. DEF SEG
540 PRINT "[Press ENTER to continue]" : LINE INPUT C$
550 REM output simple variables
560 GOSUB 1840
570 PRINT "[Press ENTER to continue]" : LINE INPUT C$
580 GOSUB 2050
590 PRINT "[Press ENTER to dump BASIC program]" : LINE INPUT C$
```

```
600 GOSUB 730
610 PRINT "Done."
620 STOP
630 REM ---
640 V = FNPEEKW(OFFS%)
650 H$ = HEX$(V)
660 H$ = RIGHT$("000"+H$,4)
670 PRINT MSG$; "-> 0x"; H$; " ="; STR$(V); "d (";HEX$(PEEK(OFFS%));
  " ";HEX$(PEEK(OFFS%+1));")";
680 IF V>4700 AND V<20000 THEN PRINT " -> ";HEX$(PEEK(V));",";HEX$(PEEK(V+1));",";
  HEX$(PEEK(V+2));",";HEX$(PEEK(V+3));",...";
690 PRINT
700 RETURN
710 REM --- dump BASIC program tokens
720 REM TXTTAB -> start of program
730 TXT% = 315
740 REM address of first line of BASIC program
750 TXT% = FNPEEKW(TXT%)
760 REM address of next line
770 NXT% = FNPEEKW(TXT%)
780 IF NXT% = 0 THEN GOTO 1260
790 REM line number
800 L% = FNPEEKW(TXT% + 2)
810 PRINT "LINE "; L%; " at "; TXT%; ":"; CHR$(9);
820 FOR I% = TXT% + 4 TO NXT% - 1
830  C% = PEEK(I%)
840  PRINT " "; MID$("0" + HEX$(C%), 1 - (C% > 15));
850 NEXT I%
860 PRINT
870 PRINT "                       "; CHR$(9);
880 J% = 0
890 FOR I% = TXT% + 4 TO NXT%-1
900  C% = PEEK(I%)
910  REM 0B: two byte OCT number
920  IF C% = &HB THEN N% = FNPEEKW(I%+1) : I% = I% + 2: PRINT " &O"; OCT$(N%); : GOTO 1210
930  REM 0C: two byte HEX number
940  IF C% = &HC THEN N% = FNPEEKW(I%+1) : I% = I% + 2: PRINT " &H"; HEX$(N%); : GOTO 1210
950  REM 0D: two byte address of line (compiled, GOTO line# becomes GOTO ADDR)
960  IF C% = &HD THEN A% = FNPEEKW(I%+1) : I% = I% + 2: PRINT " "; FNPEEKW(A% + 3) : GOTO 1210
970  REM 0E: two byte line number (not yet compiled)
980  IF C% = &HE THEN A% = FNPEEKW(I%+1) : I% = I% + 2: PRINT " "; A%; : GOTO 1210
990  REM 0F: single byte integer number 0..255
1000  IF C% = &HF THEN N% = PEEK(I%+1): I% = I% + 1: PRINT " "; N%; : GOTO 1210
1010  REM 1C: two byte integer number
1020  IF C% = &H1C THEN N% = FNPEEKW(I%+1) : I% = I% + 2: PRINT " "; N%; : GOTO 1210
1030  REM 1D: four byte single floating point number
1040  IF C% = &H1D THEN  K%=4 : GOSUB 1270: I% = I% + 4: PRINT " "; N$; : GOTO 1210
1050  REM 1E: not handled
1060  REM 1F: eight byte double floating point number
1070  IF C% = &H1F THEN K%=8 : GOSUB 1270: I% = I% + 8: PRINT " "; N$; : GOTO 1210
1080  REM escaped codes
1090  IF C% = &HFD THEN J% = C%: GOTO 1210
1100  IF C% = &HFE THEN J% = C%: GOTO 1210
1110  IF C% = &HFF THEN J% = C%: GOTO 1210
1120  REM regular command token
1130  IF J% = 0 AND C% > &H80 AND C% < (&H80 + 127) THEN
  PRINT " "; CMD$(C% - &H81); : GOTO 1210
1140  REM escaped command token
1150  IF J% = &HFD AND C% > &H80 AND C% < (&H80 + 127) THEN
  PRINT " "; XFD$(C% - &H81);""; : J% = 0: GOTO 1210
1160  IF J% = &HFE AND C% > &H80 AND C% < (&H80 + 127) THEN
  PRINT " "; XFE$(C% - &H81);""; : J% = 0: GOTO 1210
1170  IF J% = &HFF AND C% > &H80 AND C% < (&H80 + 127) THEN
  PRINT " "; XFF$(C% - &H81);""; : J% = 0: GOTO 1210
1180  REM regular characters
1190  IF C% < &H20 AND C% > 0 THEN PRINT C% - &H11; : GOTO 1210
1200  IF C% < &H20 OR C% > &H80 THEN PRINT " --";  ELSE PRINT " "; CHR$(C%); " ";
1210 NEXT I%
1220 PRINT
1230 REM address of next line
1240 TXT% = NXT%
1250 GOTO 770
1260 RETURN
1270 REM convert floating point number of K% (4 or 8) bytes
1280 REM test for zero
1290 B%=0
```

```
1300 FOR D%=1 TO K% : B%=B%+PEEK(I%+D%) : NEXT D%
1310 IF B%=0 THEN N!=0! : GOTO 1430
1320 REM biased exponent
1330 E%=PEEK(I%+K%)-129
1340 REM negative sign bit
1350 S%=1-2*((PEEK(I%+K%-1) AND &H80)\128)
1360 D!=1! : REM D!=start value for bit value: 1.0, 0.5, 0.25, ...
1370 S!=1! : REM S!=start value for sum
1380 O%=&H7F : REM mask off sign on first mantissa byte
1390 FOR D%=K%-1 TO 1 STEP -1
1400  B%=PEEK(I%+D%) AND O% : GOSUB 1750 : O%=&HFF
1410 NEXT D%
1420 N!=S%*S!*2^E%
1430 N$=STR$(N!)
1440 N$=N$+SPACE$(3+K%*3-LEN(N$))
1450 RETURN
1460 REM CMD$ single byte tokens &H81 ... &HFC = 124 items
1470 DATA "END","FOR","NEXT","DATA","INPUT","DIM","READ","LET","GOTO"
1480 DATA "RUN","IF","RESTORE","GOSUB","RETURN","REM","STOP","PRINT","CLEAR"
1490 DATA "LIST","NEW","ON","WAIT","DEF","POKE","CONT","","","OUT"
1500 DATA "LPRINT","LLIST","","WIDTH","ELSE","TRON","TROFF","SWAP","ERASE"
1510 DATA "EDIT","ERROR","RESUME","DELETE","AUTO","RENUM","DEFSTR","DEFINT"
1520 DATA "DEFSNG","DEFDBL","LINE","WHILE","WEND","CALL","","",""
1530 DATA "WRITE","OPTION","RANDOMIZE","OPEN","CLOSE","LOAD","MERGE"
1540 DATA "SAVE","COLOR","CLS","MOTOR","BSAVE","BLOAD","SOUND","BEEP"
1550 DATA "PSET","PRESET","SCREEN","KEY","LOCATE","","TO","THEN","TAB("
1560 DATA "STEP","USR","FN","SPC(","NOT","ERL","ERR","STRING$","USING"
1570 DATA "INSTR","'","VARPTR","CSRLIN","POINT","OFF","INKEY$"
1580 DATA "","","","","","","",">","=","<","+","-","*","/","^"
1590 DATA "AND","OR","XOR","EQV","IMP","MOD","\","","","","","","","",""
1600 REM XFD$ escaped tokens &HFD + &H81...&H8B = 11 items
1610 DATA "CVI","CVS","CVD","MKI$","MKS$","MKD$","","","","","EXTERR"
1620 REM XFE$ escaped tokens &HFE + &H81...&HA8 = 40 items
1630 DATA "FILES","FIELD","SYSTEM","NAME","LSET","RSET","KILL","PUT","GET"
1640 DATA "RESET","COMMON","CHAIN","DATE$","TIME$","PAINT","COM","CIRCLE"
1650 DATA "DRAW","PLAY","TIMER","ERDEV","IOCTL","CHDIR","MKDIR","RMDIR"
1660 DATA "SHELL","ENVIRON","VIEW","WINDOW","PMAP","PALETTE","LCOPY","CALLS"
1670 DATA "","","","","","LOCK","UNLOCK"
1680 REM XFF$ escaped tokens &HFF + &H81...&HA5 = 37 items
1690 DATA "LEFT$","RIGHT$","MID$","SGN","INT","ABS","SQR","RND","SIN","LOG"
1700 DATA "EXP","COS","TAN","ATN","FRE","INP","POS","LEN","STR$","VAL","ASC"
1710 DATA "CHR$","PEEK","SPACE$","OCT$","HEX$","LPOS","CINT","CSNG","CDBL"
1720 DATA "FIX","PEN","STICK","STRIG","EOF","LOC","LOF"
1730 REM -------------------
1740 REM add value of each bit in byte J% to sum S!
1750 M%=&H80 : REM mask
1760 FOR F%=0 TO 7
1770  IF (B% AND M%) THEN S!=S!+D!
1780  REM shift mask
1790  M%=M%\2
1800  REM next bit value
1810  D!=D!/2!
1820 NEXT F%
1830 RETURN
1840 REM ----- output of simple variables
1850 PRINT "Simple Variables:"
1860 OFFS% = 1184 : ADR% = FNPEEKW(OFFS%) : REM ARYTAB
1870 OFFS% = 1182 : TXT% = FNPEEKW(OFFS%)
1880 T%=PEEK(TXT%) : TXT%=TXT%+1 : REM type 2,3,4,8
1890 C%=PEEK(TXT%) : N$=CHR$(C%) : TXT%=TXT%+1 : REM 1st character of name
1900 IF C% AND &H80 THEN N$="fn"+CHR$(C% AND &H7F) : REM function
1910 C%=PEEK(TXT%) : TXT%=TXT%+1 : REM 2nd character of name or zero
1920 IF C%<>0 THEN N$=N$+CHR$(C%)
1930 N%=PEEK(TXT%) : TXT%=TXT%+1 : REM number of extra chars in name
1940 FOR I%=1 TO N% : C%=PEEK(TXT%) AND &H7F : N$=N$+CHR$(C%) : TXT%=TXT%+1 : NEXT I%
1950 IF T%=2 THEN N$=N$+"%":N$="INTEGER "+CHR$(9)+N$
1960 IF T%=3 THEN N$=N$+"$":N$="STRING  "+CHR$(9)+N$
1970 IF T%=4 THEN N$=N$+"!":N$="SINGLE  "+CHR$(9)+N$
1980 IF T%=8 THEN N$=N$+"#":N$="DOUBLE  "+CHR$(9)+N$
1990 IF MID$(N$,10,2)="fn" THEN N$=N$+"   (function)"
2000 PRINT T%;N$
2010 TXT%=TXT%+T% : REM advance over data to next variable
2020 IF TXT%<ADR% THEN GOTO 1880
2030 RETURN
2040 REM ----- output of array variable
2050 PRINT "Array Variables:"
```

```
2060 OFFS% = 1186 : ADR% = FNPEEKW(OFFS%) : REM below STREND?
2070 OFFS% = 1184 : TXT% = FNPEEKW(OFFS%)
2080 T%=PEEK(TXT%) : TXT%=TXT%+1 : REM type 2,3,4,8
2090 C%=PEEK(TXT%) : N$=CHR$(C%) : TXT%=TXT%+1 : REM 1st character of name
2100 C%=PEEK(TXT%) : TXT%=TXT%+1 : REM 2nd character of name or zero
2110 IF C%<>0 THEN N$=N$+CHR$(C%)
2120 N%=PEEK(TXT%) : TXT%=TXT%+1 : REM number of extra chars in name
2130 FOR I%=1 TO N% : C%=PEEK(TXT%) AND &H7F : N$=N$+CHR$(C%) : TXT%=TXT%+1 : NEXT I%
2140 I% = FNPEEKW(TXT%) : TXT%=TXT%+2 : REM SIZE bytes follow
2150 N% = PEEK(TXT%) : REM NDIM
2160 IF T%=2 THEN N$=N$+"%":N$="INTEGER "+CHR$(9)+N$
2170 IF T%=3 THEN N$=N$+"$":N$="STRING  "+CHR$(9)+N$
2180 IF T%=4 THEN N$=N$+"!":N$="SINGLE  "+CHR$(9)+N$
2190 IF T%=8 THEN N$=N$+"#":N$="DOUBLE  "+CHR$(9)+N$
2200 PRINT T%;N$;" (";
2210 FOR I=TXT%+N%*2-1 TO TXT%+1 STEP -2 : REM right to left
2220  PRINT FNPEEKW(I);
2230  IF I>TXT%+1 THEN PRINT ",";
2240 NEXT I
2250 PRINT ")   ";N%;"DIMENSION(S)";I%;"BYTES INCL. NDIM,DIM..."
2260 TXT%=TXT%+I% : REM advance over data to next array
2270 IF TXT%<ADR% THEN GOTO 2080
2280 RETURN
```

Listing 5: Program GWMEM walks through its own memory. Addresses are valid for GWBASIC version 3.23.

```
10 REM
20 REM Microsoft BASIC 5.28 memory dumper
30 REM Addresses valid for MBASIC.EXE
40 REM size: 31744 bytes
50 REM
60 REM Martin Hepperle, 2020
70 REM
80 REM Make sure that segment address is default
90 DEF SEG
100 DIM ARY(128)
110 DIM CMD$(126),XFF$(81)
120 REM get a word at A -> FN becomes a simple variable, bit 7 in 1st character set
130 DEF FNPEEKW(A)=PEEK(A)+256*PEEK(A+1)
140 RESET
150 REM Read single byte tokens
160 FOR I% = 0 TO 125 : READ CMD$(I%) : NEXT I%
170 REM Read escaped tokens
180 FOR I% = 0 TO 80 : READ XFF$(I%) : NEXT I%
190 PRINT "Start offset for HEX dump";
200 INPUT S
210 E=S+4*64-1
220 L$ = ""
230 N=16
240 PRINT "MEMORY FROM";S;"TO";E;
250 REM PRINT RIGHT$("000"+HEX$(S),4);": ";
260 FOR I = S TO E
270  IF N = 16 THEN PRINT " "; L$ : N = 0 : L$ = "" : PRINT RIGHT$("000"+HEX$(I),4);": ";
280  C% = PEEK(I)
290  IF C% > 31 AND C%<128 THEN C$ = CHR$(C%) ELSE C$ = "."
300  L$ = L$ + C$
310  PRINT MID$("0" + HEX$(C%), 1 - (C% > 15)); " ";
320  N=N+1
330 NEXT I
340 IF N>0 THEN FOR I=1 TO 16-N : PRINT "   "; : NEXT I : PRINT " ";L$
350 PRINT "[Press ENTER to continue]" : LINE INPUT C$
360 PRINT "Variables:"
370 MSG$ = "TOPMEM": OFFS% = 348: GOSUB 620 : REM O.K. -> top of stack
380 MSG$ = "MAXMEM": OFFS% = 1157: GOSUB 620 : REM O.K. -> &HFFFE data seg size
390 MSG$ = "MEMSIZ": OFFS% = 1159: GOSUB 620 : REM O.K. -> &HFDFC max. offset used
400 MSG$ = "FRETOP": OFFS% = 1196: GOSUB 620 : REM O.K. ->
410 REM MSG$ = "DEVTBL": OFFS% = 358: GOSUB 630 : REM ???
420 MSG$ = "STREND": OFFS% = 1244: GOSUB 620 : REM O.K. -> lower end of string vars
430 MSG$ = "ARYTAB": OFFS% = 1242: GOSUB 620 : REM O.K. -> array variables
440 MSG$ = "ARYTA2": OFFS% = 1483: GOSUB 620: REM O.K. same as ARYTAB
450 MSG$ = "VARTAB": OFFS% = 1240: GOSUB 620 : REM O.K. -> simple variables
460 MSG$ = "TXTTAB": OFFS% = 352: GOSUB 620 : REM O.K. -> program TEXT
470 MSG$ = "DATPTR": OFFS% = 1246: GOSUB 620 : REM O.K. -> initial DATA: 0 byte before TEXT
480 MSG$ = "AUTINC": OFFS% = 1215: GOSUB 620 : REM O.K. AUTO line # increment
```

```
490 MSG$ = "USRTAB": OFFS% = 285: GOSUB 620 : REM O.K. -> USR0 to USR9 functions
500 MSG$ = "DEFTBL": OFFS% = 1248: GOSUB 620 : REM O.K. -> 26 default types, 4,4 ...
510 MSG$ = "SAVSEG": OFFS% = 1230: GOSUB 620 : REM O.K. DEF SEG
520 PRINT "[Press ENTER to continue]" : LINE INPUT C$
530 REM output simple variables
540 GOSUB 1720
550 PRINT "[Press ENTER to continue]" : LINE INPUT C$
560 GOSUB 1930
570 PRINT "[Press ENTER to dump BASIC program]" : LINE INPUT C$
580 GOSUB 710
590 PRINT "Done."
600 STOP
610 REM ---
620 V = FNPEEKW(OFFS%)
630 H$ = HEX$(V)
640 H$ = RIGHT$("000"+H$,4)
650 PRINT MSG$; "-> 0x"; H$; " ="; STR$(V); "d (";HEX$(PEEK(OFFS%));" ";
  HEX$(PEEK(OFFS%+1));")";
660 IF V>4700 AND V<20000 THEN PRINT " -> ";HEX$(PEEK(V));",";HEX$(PEEK(V+1));",";
  HEX$(PEEK(V+2));",";HEX$(PEEK(V+3));",...";
670 PRINT
680 RETURN
690 REM --- dump BASIC program tokens
700 REM TXTTAB -> start of program
710 TXT% = 352
720 REM address of first line of BASIC program
730 TXT% = FNPEEKW(TXT%)
740 REM address of next line
750 NXT% = FNPEEKW(TXT%)
760 IF NXT% = 0 THEN GOTO 1200
770 REM line number
780 L% = FNPEEKW(TXT% + 2)
790 PRINT "LINE "; L%; " at "; TXT%; ":"; CHR$(9);
800 FOR I% = TXT% + 4 TO NXT% - 1
810  C% = PEEK(I%)
820  PRINT " "; MID$("0" + HEX$(C%), 1 - (C% > 15));
830 NEXT I%
840 PRINT
850 PRINT "                        "; CHR$(9);
860 J% = 0
870 FOR I% = TXT% + 4 TO NXT%-1
880  C% = PEEK(I%)
890  REM 0B: two byte OCT number
900  IF C% = &HB THEN N% = FNPEEKW(I%+1) : I% = I% + 2 : PRINT " &O"; OCT$(N%); : GOTO 1150
910  REM 0C: two byte HEX number
920  IF C% = &HC THEN N% = FNPEEKW(I%+1) : I% = I% + 2 : PRINT " &H"; HEX$(N%); : GOTO 1150
930  REM 0D: two byte address of line (compiled, GOTO line# becomes GOTO ADDR)
940  IF C% = &HD THEN A% = FNPEEKW(I%+1) : I% = I% + 2 :
  PRINT " ";PEEK(A% + 3) + 256 * PEEK(A% + 4) : GOTO 1150
950  REM 0E: two byte line number (not yet compiled)
960  IF C% = &HE THEN A% = FNPEEKW(I%+1) : I% = I% + 2 : PRINT " "; A%; : GOTO 1150
970  REM 0F: single byte integer number 0..255
980  IF C% = &HF THEN N% = PEEK(I%+1) : I% = I% + 1 : PRINT " "; N%; : GOTO 1150
990  REM 1C: two byte integer number
1000  IF C% = &H1C THEN N% = FNPEEKW(I%+1) : I% = I% + 2 : PRINT " "; N%; : GOTO 1150
1010  REM 1D: four byte single floating point number
1020  IF C% = &H1D THEN  K%=4 : GOSUB 1210 : I% = I% + 4 : PRINT " "; N$; : GOTO 1150
1030  REM 1E: not handled
1040  REM 1F: eight byte double floating point number
1050  IF C% = &H1F THEN K%=8 : GOSUB 1210: I% = I% + 8: PRINT " "; N$; : GOTO 1150
1060  REM escaped codes
1070  IF C% = &HFF THEN J% = C% : GOTO 1150
1080  REM regular command token
1090  IF J% = 0 AND C% > &H80 AND C% < (&H80 + 127) THEN
  PRINT " "; CMD$(C% - &H81); : GOTO 1150
1100  REM escaped command token
1110  IF J% = &HFF AND C% > &H80 AND C% < (&H80 + 127) THEN
  PRINT " "; XFF$(C% - &H81);""; : J% = 0: GOTO 1150
1120  REM regular characters
1130  IF C% < &H20 AND C% > 0 THEN PRINT C% - &H11; : GOTO 1150
1140  IF C% < &H20 OR C% > &H80 THEN PRINT " --";  ELSE PRINT " "; CHR$(C%); " ";
1150 NEXT I%
1160 PRINT
1170 REM address of next line
1180 TXT% = NXT%
1190 GOTO 750
```

```
1200 RETURN
1210 REM convert floating point number of K% (4 or 8) bytes
1220 REM test for zero
1230 B%=0
1240 FOR D%=1 TO K% : B%=B%+PEEK(I%+D%) : NEXT D%
1250 IF B%=0 THEN N!=0! : GOTO 1370
1260 REM biased exponent
1270 E%=PEEK(I%+K%)-129
1280 REM negative sign bit
1290 S%=1-2*((PEEK(I%+K%-1) AND &H80)\128)
1300 D!=1! : REM D!=start value for bit value: 1.0, 0.5, 0.25, ...
1310 S!=1! : REM S!=start value for sum
1320 O%=&H7F : REM mask off sign on first mantissa byte
1330 FOR D%=K%-1 TO 1 STEP -1
1340  B%=PEEK(I%+D%) AND O% : GOSUB 1630 : O%=&HFF
1350 NEXT D%
1360 N!=S%*S!*2^E%
1370 N$=STR$(N!)
1380 N$=N$+SPACE$(3+K%*3-LEN(N$))
1390 RETURN
1400 REM CMD$ single byte tokens &H81 ... &HFC = 128 items
1410 DATA "END","FOR","NEXT","DATA","INPUT","DIM","READ","LET","GOTO"
1420 DATA "RUN","IF","RESTORE","GOSUB","RETURN","REM","STOP","PRINT","CLEAR"
1430 DATA "LIST","NEW","ON","NULL","WAIT","DEF","POKE","CONT","","","OUT"
1440 DATA "LPRINT","LLIST","","WIDTH","ELSE","TRON","TROFF","SWAP","ERASE"
1450 DATA "EDIT","ERROR","RESUME","DELETE","AUTO","RENUM","DEFSTR","DEFINT"
1460 DATA "DEFSNG","DEFDBL","LINE","BLOAD","BSAVE","WHILE","WEND","CALL"
1470 DATA "WRITE","COMMON","CHAIN","OPTION","RANDOMIZE","CALLS","SYSTEM"
1480 DATA "","OPEN","FIELD","GET","PUT","CLOSE","LOAD","MERGE","FILES"
1490 DATA "NAME","KILL","LSET","RSET","SAVE","RESET","","TO","THEN","TAB("
1500 DATA "STEP","USR","FN","SPC(","NOT","ERL","ERR","STRING$","USING"
1510 DATA "INSTR","","VARPTR","INKEY$","","","","","","","","","","",""
1520 DATA "","","","","","",">","=","<","+","-","*","/","^"
1530 DATA "AND","OR","XOR","EQV","IMP","MOD","\",""
1540 REM XFF$ escaped tokens &HFF + &H81...&HD1 = 81 items
1550 DATA "LEFT$","RIGHT$","MID$","SGN","INT","ABS","SQR","RND","SIN","LOG"
1560 DATA "EXP","COS","TAN","ATN","FRE","INP","POS","LEN","STR$","VAL","ASC"
1570 DATA "CHR$","PEEK","SPACE$","OCT$","HEX$","LPOS","CINT","CSNG","CDBL"
1580 DATA "FIX","","","","","","","","","","","","CVI","CVS","CVD","","EOF"
1590 DATA "LOC","LOF","MKI$","MKS$","MKD$","","","","","","","","","","",""
1600 DATA "","","","","","","","","","","","","","","","","DATE","TIME"
1610 REM -------------------
1620 REM add value of each bit in byte J% to sum S!
1630 M%=&H80 : REM mask
1640 FOR F%=0 TO 7
1650  IF (B% AND M%) THEN S!=S!+D!
1660  REM shift mask
1670  M%=M%\2
1680  REM next bit value
1690  D!=D!/2!
1700 NEXT F%
1710 RETURN
1720 REM ----- output of simple variables
1730 PRINT "Simple Variables:"
1740 OFFS% = 1242 : ADR% = FNPEEKW(OFFS%) : REM ARYTAB
1750 OFFS% = 1240 : TXT% = FNPEEKW(OFFS%) : REM VARTAB
1760 T%=PEEK(TXT%) : TXT%=TXT%+1 : REM type 2,3,4,8
1770 C%=PEEK(TXT%) : N$=CHR$(C%) : TXT%=TXT%+1 : REM 1st character of name
1780 IF C% AND &H80 THEN N$="fn"+CHR$(C% AND &H7F) : REM function
1790 C%=PEEK(TXT%) : TXT%=TXT%+1 : REM 2nd character of name or zero
1800 IF C%<>0 THEN N$=N$+CHR$(C%)
1810 N%=PEEK(TXT%) : TXT%=TXT%+1 : REM number of extra chars in name
1820 FOR I%=1 TO N% : C%=PEEK(TXT%) AND &H7F : N$=N$+CHR$(C%) : TXT%=TXT%+1 : NEXT I%
1830 IF T%=2 THEN N$=N$+"%":N$="INTEGER "+CHR$(9)+N$
1840 IF T%=3 THEN N$=N$+"$":N$="STRING  "+CHR$(9)+N$
1850 IF T%=4 THEN N$=N$+"!":N$="SINGLE  "+CHR$(9)+N$
1860 IF T%=8 THEN N$=N$+"#":N$="DOUBLE  "+CHR$(9)+N$
1870 IF MID$(N$,10,2)="fn" THEN N$=N$+"   (function)"
1880 PRINT T%;N$
1890 TXT%=TXT%+T% : REM advance over data to next variable
1900 IF TXT%<ADR% THEN GOTO 1760
1910 RETURN
1920 REM ----- output of array variable
1930 PRINT "Array Variables:"
1940 OFFS% = 1244 : ADR% = FNPEEKW(OFFS%) : REM STREND
1950 OFFS% = 1242 : TXT% = FNPEEKW(OFFS%) : REM ARYTAB
```

```
1960 T%=PEEK(TXT%) : TXT%=TXT%+1 : REM type 2,3,4,8
1970 C%=PEEK(TXT%) : N$=CHR$(C%) : TXT%=TXT%+1 : REM 1st character of name
1980 C%=PEEK(TXT%) : TXT%=TXT%+1 : REM 2nd character of name or zero
1990 IF C%<>0 THEN N$=N$+CHR$(C%)
2000 N%=PEEK(TXT%) : TXT%=TXT%+1 : REM number of extra chars in name
2010 FOR I%=1 TO N% : C%=PEEK(TXT%) AND &H7F : N$=N$+CHR$(C%) : TXT%=TXT%+1 : NEXT I%
2020 I% = FNPEEKW(TXT%) : TXT%=TXT%+2 : REM SIZE bytes follow
2030 N% = PEEK(TXT%) : REM NDIM
2040 IF T%=2 THEN N$=N$+"%":N$="INTEGER "+CHR$(9)+N$
2050 IF T%=3 THEN N$=N$+"$":N$="STRING  "+CHR$(9)+N$
2060 IF T%=4 THEN N$=N$+"!":N$="SINGLE  "+CHR$(9)+N$
2070 IF T%=8 THEN N$=N$+"#":N$="DOUBLE  "+CHR$(9)+N$
2080 PRINT T%;N$;" (";
2090 FOR I=TXT%+N%*2-1 TO TXT%+1 STEP -2 : REM right to left
2100  PRINT FNPEEKW(I);
2110  IF I>TXT%+1 THEN PRINT ",";
2120 NEXT I
2130 PRINT ")   ";N%;"DIMENSION(S)";I%;"BYTES INCL. NDIM,DIM..."
2140 TXT%=TXT%+I% : REM advance over data to next array
2150 IF TXT%<ADR% THEN GOTO 1960
2160 RETURN
```

Listing 6: Program BASMEM walks through its own memory. Addresses are valid for MBASIC version 5.28.

BASIC Programs

BASIC programs are stored line by line in a mix of tokens and ASCII text. All keywords and functions are tokenized i.e. translated into a byte code. Also, numeric constants are encoded in binary form and line numbers are translated to addresses. This shortens the program code and accelerates its interpretation. When the program is listed, all tokens are translated back into their human readable form. The token numbers vary with major version of the BASIC interpreter. Also, depending on the platform, new tokens may have been added or removed to the same version of the interpreter.

Tokens can be single bytes or a combination of an escape code and the following byte. Early MBASIC versions used the single escape code `&HFF`, later this scheme was extended to include `&HFD` and `&HFE` escape codes.

The address of the first program line is stored in `TXTTAB`.

Table 4: Structure of a program line.

Offset	Length	Description
0	2	Address of the next line, so that the current line can be quickly skipped. If this address is zero, there is no next line: the end of the program is reached.
2	2	Line number (binary).
4	...	Program line, a zero token marks the end of the line. Note that zero bytes may occur inside the line e.g. as part of a numeric constant.

```
LINE  750  at  7486 :
4E 58 54 25 20 E7 20 FF 97 28 54 58 54 25 29 20 E9 20 1C 00 01 20 EB 20 FF 97 28 54 58 54 25 E9 12 29 00
N  X  T  %     =     PEEK  (  T  X  T  %  )     +     <  256 >     *     PEEK  (  T  X  T  %  +  1  )  EOL

LINE  760  at  7525 :
8B 20 4E 58 54 25 20 E7 20 11 20 CD    20 89    20 0E CE 04 00
IF    N  X  T  %     =     0     THEN     GOTO     < 1230 > EOL

LINE  770  at  7548 :
8F 20 6C 69 6E 65 20 6E 75 6D 62 65 72 00
REM   l  i  n  e     n  u  m  b  e  r  EOL

...
```

```
LINE  2290  at  14185 :  defines a FN function
97 20 D1 46 55 4E 28 41 29 E7 28 FF 97 28 41 29 E9 1C 00 01 EB FF 97 28 41 E9 12 29 00
DEF   FN F  U  N  (  A  )  =  (  PEEK  (  A  )  +  < 256 >  *  PEEK  (  A  +  1  )  EOL
```

Listing 7: Some tokenized program lines, omitting leading address and line number.

Table 5: Tokens used in the example lines.

Token	Translation
00	end of line (EOL)
11	constant 0
12	constant 1
1C	prefix for 2-byte integer
0E	prefix for 2-byte integer
20 … 7E	ASCII characters, preserving spaces
E7	= (assignment or comparison)
E9	+ (addition)
EB	* (multiplication)
89	GOTO
8B	IF
8F	REM
97	DEF (may be followed by SEG or FN)
CD	THEN
D1	FN
FF	prefix for next byte in two-byte sequence
97	PEEK, two-byte sequence

Table 6: Location of key variables in various versions of Microsoft BASIC.

Version	Version	Year	Free	USRTAB	DAYSPM	TOPMEM	TXTTAB	BUF	SAVSEG	VARTAB	ARYTAB	STREND	DATPTR	DEFTBL	ARYTA2
BASIC-86 Rev. 5.21	86-DOS	1981	62111	279	n.a.	313	317	790	1136	1144	1146	1148	1150	1152	1387
BASIC-86 Rev. 5.27	MS-DOS	1982	62025	284	321	338	342	861	1210	1218	1220	1222	1224	1226	1461
Compaq-BASIC 1.12	Compaq	1982	62762	284	322	339	343	810	1159	1167	1169	1171	1173	1175	1410
Microsoft BASIC 5.28	MS-DOS	1983	62003	285	331	348	352	883	1232	1240	1242	1244	1246	1248	1483
GW-BASIC 1.12.04	Corona/Sperry	1984	62003	285	331	348	352	883	1181	1240	1242	1244	1246	1248	1483
BASIC 2.02/01.01.00	Tandy	1984	60965	280	310	327	331	841	1189	1197	1199	1201	1203	1205	1440
GW-BASIC 2.0/1.0	Olivetti	1983	61098	280	310	327	331	923	1272	1280	1282	1284	1286	1288	1523
GW-BASIC 2.01/1.02	Olivetti	1984	62093	248	278	295	299	828	1177	1185	1187	1189	1191	1193	1428
GW-BASIC 2.02	MS-DOS	1984	61734	248	278	295	299	808	1157	1165	1167	1169	1171	1173	1408
GW-BASIC 2.02	Bondwell	1984	61734	248	278	295	299	808	1157	1165	1167	1169	1171	1173	1408
GW-BASIC 2.02/V2.02	Commodore	1984	62194	248	278	295	299	808	1157	1165	1167	1169	1171	1173	1408
GW-BASIC 2.02/2D	Epson	1985	62053	248	278	295	299	808	1157	1165	1167	1169	1171	1173	1408
GW-BASIC 3.10/3.13	Zenith	1985	60218	248	278	295	299	808	1157	1165	1167	1169	1171	1173	1408
GW-BASIC 3.11/3.11.02	Cordata	1985	62109	248	278	295	299	808	1157	1165	1167	1169	1171	1173	1408
GW-BASIC 3.21	IBM	1987	62165	248	278	295	299	808	1157	1165	1167	1169	1171	1173	1408
GW-BASIC 3.20	Tandy	1986	59981	296	326	343	347	858	1207	1215	1217	1219	1221	1223	1458
GW-BASIC 3.20	MS-DOS	1986	60332	264	294	311	315	826	1175	1183	1185	1187	1189	1191	1426
GW-BASIC 3.20/3.16	Olivetti	1986	59834	264	294	311	315	826	1175	856	858	860	1189	1191	1185
GW-BASIC 3.22/3.29	Olivetti	1987	59834	264	294	311	315	825	1174	1182	1184	1186	1188	1190	1425
GW-BASIC 3.22	MS-DOS	1987	60300	264	294	311	315	825	1174	1182	1184	1186	1188	1190	1425
GW-BASIC 3.23	MS-DOS	1988	60300	264	294	311	315	825	1174	1182	1184	1186	1188	1190	1425

The rows shaded in light blue indicate groups with similar or identical memory layouts.
The versions labeled “MS-DOS” are generic Microsoft versions without OEM branding.