Beyond Freedom and Programming with Kangaroo

or

The Harsh Realities of Your Environment

Table of Contents

Raison d'Etre

CHAPTER 1

We will not concern ourselves here with techniques, algorithms, or other sensical matters usually associated with treatises on BASIC programs Instead we plan to demonstrate the metamorphic processes involved in the evolution of that entity we have come to know as 'MY PROGRAM'. Let us regard the following typical BASIC program.

```
BASIC PROGRAM -- 'MAGIC'

 10 for I=1 to 10
 20 input A$
 30 gosub 90
 40 input X
 50 gosub 110
 60 disp A$;X
 70 next i
 80 end
 90 A$='froggie'
100 return
110 X=pi
120 return
```

As the astitute programmer is typing the above characters into the display of Kamgaroo, he or she is no doubt pondering the infinite possibilities of the programs internal existence. To this query we devote this document.

Dialogue	CHAPTER 2

Joey:
Procrastinating programs, Roo-man! Why does it take so long for the cursor to reappear after pressing the return key? What is going on?

Roo-man:
During the interval from pressing the return key to the return of the curser many things are happening, Joey. First the M.O.(mode of operation) must be determined (calculator vs. program). Next the machine attempts to interpret what appears in the display into a format which the machine can utilize. This act of interpretation is called Parsing. Let us take an example. The first line of the program 'MAGIC' is:

10 for I=1 to 10

The first obvious hurdle is recognition and classification of the elements of the line. There are classes and subclasses to worry about also. For example: does a succession of digits imply a line number or a numeric constant? If a letter is seen is this a variable or the beginning of a command? If the letter is to be treated as a variable should it be a simple numeric variable, an array variable or a string variable? If the letter is not a variable but is the beginning of a word or a command how do we determine what letter ends the word or command?

Joey:
Parapalegic parsers, Roo-man! What is the name of this masked entity that recognizes and classifies these elements?

Roo-man:
We call it the SCANNER! The system scanner grabs each element in the line and replaces it with a token.

Joey:
Scuzzy scanners, Roo-man! What is a token?

Roo-man:
To answer this question we cite an example from the Miller Analogy Test, January 1968:

word:language

token:kangaroocese

To be more precise the set of tokens form a 1-1 correspondence with the set of all elements in our BASIC language. Each token is a one byte value that internally represents each BASIC keyword.

Joey:
Do the tokens appear in the same order as the keywords in the BASIC statement?

Roo-man:
Unfortunately, this is not the case. If it were, we would need only a scanner to interpret the lines of input and the whole process of Parsing would be that much quicker. Instead the lines are Parsed into RPN (reverse Polish notation) and stacked in appropriate order.

Joey:
Is that all that must be done? Are the tokens executed on the R12 stack?

Roo-man:
NO and NO! We are getting way ahead of ourselves by talking about execution but to clearify this fact, the tokens are NEVER executed from the R12 stack. By the way how did you know about the R12 stack?

Joey:
I read the Capricorn Assembly Rom Manual. Is that okay?

Roo-man:
One must always read the Assembly Rom Manual with caution and pragmatism. but we are far afield from the topic at hand. Let us hop back. It is a fact that the tokens are stacked in executible order on the R12 stack. Yet they are like a man without a home. They have no roof over their heads, no place to hang their hats, no faithful servent to deliever slippers, pipe and paper. In short, neuters tokens.

Joey:
How incredibly tragic! Is there a solution?

Roo-man:
Ah, indeed yes. The final step to our procedure is to insert this stream of tokens, stacked on R12, into their appropriate location in the memory. This is done by locating the correct sequential position for the line number. Once this is found, the line number and the tokenized line are inserted into memory at this location. Let us contrast the external and internal existences. Recall that the first line of 'MAGIC' is

10 for I=1 to 10

Internally this looks like

line#	size	for	ftadr	var I	intcon	one
10 00	0F	8C	11	20 49	1A	01 00 00

stosv	intcon	ten	to	eol
08	1A	10 00 00	9F	0E

Notice that each token stream is preceeded by two bytes for the BCD line number and one byte which represents the length of the token stream including the end-of-line token, 0E.

Joey:
But line 10 is the first line of the program. What sequence exists that allows insertion of the number 10?

Roo-man:
Good question! In the beginning Kangaroo created a user workspace called the 'workfile'. The original structure of the 'workfile' is as follows. The first 10 bytes are zeros follo...

Joey:
What are the 10 zeros for?

Roo-man:
Their reason for being will be defined later. Please be patient. As I was saying the original structure is 10 bytes of zeros followed by the internal end line, which has a line number of A999.

Joey:
But A999 is an illegal line number.

Roo-man:
Indeed, yes. Since the average programmer cannot construct a line number greater than 9999 the internal end line can never be subverted and always remains as the last line in our program. Hence our sequence is always present and never invalid!

Joey:
I see. Since the first line number will always be less than A999, the first line will always be inserted correctly. Does it go before or after the 10 bytes of zeros?

Roo-man:
The first 10 bytes of most files are reserved for a very special occasion. Our first line will therefore be inserted after the first 10 bytes. Let's look at the before and after shot of the 'workfile'.

Before:

10 Bytes of Zeros

00 00 00 00 00 00 00 00 00 00

Endline

99 A9 02 8A 0E

After:

10 Bytes of Zeros

00 00 00 00 00 00 00 00 00 00

Line 10

10 00 0F 8C 11 20 49 1A 01 00

00 08 1A 10 00 00 9F 0E

Endline

99 A9 02 8A 0E

Joey:
Is there only one file in Kangaroo?

Roo-man:
No, there may be many files.

Joey:
Well, if there can be many files, how is the correct file located?

Roo-man:
Kangaroo maintains a directory. Each directory entry contains pertinent information about a file. For example, after we have typed in line 10, the directory entry for the 'workfile' loo...

Joey:
I thought you said the programs name was 'MAGIC'?

Roo-man:
Is this going to habitual? Let's regress for a moment. Every time a Programmer feels the need to be creative, Kangaroo provides a workspace. The command EDIT BASIC signifies to Kangaroo that you have your thinking cap tied on and would like to start programming. If your intention is to create, say a letter to your mother or memos to yourself, the command EDIT TEXT would be used. In both cases Kangaroo provides a 'workfile' of the

respective type and any lines constructed by you will be inserted into that 'workfile'. Now, if we may, let us look at the directory entry for the 'workfile' after line 10 has been inserted.

```
85   93            :location of file in memory
00   21            :size of file
42                 :type of file (42-->Basic)
FE                 :type of access
9B   11   91   DE  :date of creation
workfile           :name
```

Now to appease your curiousity type RENAME TO 'MAGIC' on the Kangaroo

Joey:
R E N M A E T O ' M A G I C ' . Why did I get this error syntax?

Roo-man:
The machine does not understand Scottish. Try it again.

Joey:
I don't understand Scottish either!

Roo-man:
TRY IT AGAIN PLEASE!

Joey:
Geez, something wrong? Okay, okay, never mind. R E N A M E T O ' M A G I C '.

Roo-man:
Now examine the directory.

```
85   9B
00   21
42
FE
9B   11   91   DE
MAGIC
```

If there are no further questions at the moment would you be so kind as to type the rest of our sample program into Kangaroo.

Joey:
I would like to ask one more question before we exit this chapter.

Roo-man:
Go ahead.

Joey:
What is that weird Date of Creation for the program? What does 9B 11 91 DE stand for? How am I supposed to tell when the program was created? How

many leaves in a forest? How many grains of sand on a beach? How ...

Roo-man:
Yo! Please constrain yourself! I see the necessity for a short tale about the internal date of creation. Don't stare, but see that rather stocky fellow with the beard and pleasent expression on his face. If you look a little closer yo.. DON'T STARE! you will notice the rather unbalanced gleam in his eyes. Believe it or not he was once an exact look alike of Charles Atlas. That was until the day he attempted to explain, to a novice such as yourself, the true meaning of the IDC. Ah me, the painful memories of that horrid afternoon when the Great Horned Roo so mercilessly punished he who naively sought to explain the unexplainable.

Joey:
Gosh Roo-man, I never would have drea... Roo-man? Oh Roo-man? Roo-man!

Roo-man:
Excuse me, but I feel the need of a short interlude. Please continue entering the program, I shall return.

Interlude	CHAPTER 3

Roo-man:
Well, I see you have completed your task. Shall we examine the program as it resides internally?

Joey:
Sure!

Roo-man:
Such enthusiasm! It is so satisfying to an educator, such as myself, when a student demonstrates a thirst for knowledge.

Joey:
I don't know about knowledge, but I sure would like a glass of milk.

Roo-man:
Next interlude. First the program as it appears in memory.

10 bytes of zeros

00	00	00	00	00	00	00	00	00	00

Program

10	00	0F	8C	11	20	49	1A	81	00
		00	00	1A	10	00	00	9F	0E
20	00	07	5F	13	20	41	E5	19	0E
30	00	04	58	00	00	0E			
40	00	07	5F	11	20	53	DD	19	0E
50	00	04	58	00	01	0E			
60	00	0B	56	03	20	41	A3	01	20
		53	E7	A2	0E				
70	00	05	11	20	49	8F	0E		
80	00	02	8A	0E					

90	00	0E	13	20	41	96	07	66	72
		6F	67	67	69	65	07	0E	
00	01	02	71	0E					
10	01	06	11	20	58	C9	08	0E	
20	01	02	71	0E					

Endline

99	A9	02	8A	0E

Roo-man:
As you can see, the line numbers form an increasing sequence approaching to A999. Did you notice the similarities between line 80 and line A999.

Joey:
yep

Roo-man:
What are you doing?

Joey:

Roo-man:

Joey:

Roo-man:
Mon Dieu!

Joey
Huh?

Roo-man:
Before we proceed with the phases of program execution, I must insist that you clean off the keyboard. I shall return.

More Dialogue	CHAPTER 5

Roo-man:
It is time to discuss the initial 10 bytes which preceed each lined file in memory. As you recall, at creation time these bytes are initialized to zero. The term we use to define this collection of bytes is the Program Control Block or more commonly called the PCB. The structure of the PCB is...

Joey:
Aren't PCB's those bad chemicals found around the Love Canal near Buffalo, New York?

Roo-man:
(is this a nightmare?) No, actually, you are refering to PCP's. As I was saying, the structure of the PCB is 5 blocks of 2 byte values. These are

bytes	name	description
0/1	P.LEN	length of program and PCB
2/3	P.FLEN	length of variable ptr area
4/5	P.CLEN	spare block
6/7	P.ELEN	length of environment-ECB
8/9	P.SPAR	spare block

Joey:
Why is this PCB necessary? Seems to me like a waste of memory.

Roo-man:
As time goes by, the value of the PCB will become more clear. Suffice it to say, that with out the information given in the PCB, program execution in Kangaroo would be considerably slower.

Joey:
Well, when are these values placed in the PCB?

Roo-man:
Thought you'd never ask. When you push the RUN key or type RUN 'MAGIC', a series of events takes place. First the program is Pointer Allocated. This means that variable names and BCD line numbers are replaced with relative pointers; relative to the beginning of the program that is. For example, it is much easier for the machine to obey the instruction GOTO (a location 30 bytes relative to the beginning of the program) rather than

GOTO (the line with the BCD line number 100). In the case of variable it is an absolute necessity that the name be replaced with a pointer to information about the variable. Take the variable name I. Is this a string, array or simple numeric variable?

Joey:
Well, I guess it is a numeric variable.

Roo-man:
Why do you say that?

Joey:
It doesn't have a '$' or a '(' after it.

Roo-man:
Oh? Would you please look at the variable name for A$ in line 20.

20 00 07 3F 13 20 41 E5 19 0E

In particular the two bytes 20 41. The ASCII representation for these two bytes is 'A '. Now, if you will, would you mind classifying this variable.

Joey:
Okay, okay. You don't have to rub it in. But how do you tell what type of variable it is?

Roo-man:
That's eas..

Joey:
But how do you know if it is a function or just a variable?

Roo-man:
I was trying to tel..

Joey:
But how about whether its a REAL or an INTEGER or, or, or..

Roo-man:
FERMEZ LA BOUCHE, si vous plait. Bien. Now, as I was about to explain, every variable has ALL of its necessary characteristic information condensed into a 2 byte quantity called its Name Form. Can your think any other information about the variable that would be useful?

Joey:
How about the value of the variable or where the value of the variable can be found?

Roo-man:
Not bad! For simple numeric variables, the only information necessary is the internal Name Form and the location of the variables value. String

variables require one more piece of information, that being the maximum allocated length of the string. Whereas array variables require the maximum row and column dimensions along with the total length allocated for the array elements values. Each variable has its own block of information in the Variable Pointer Area. If you would like more detailed information on the VPA you may acquire the document on Program Pointer Allocation.

Joey:
Yea, okay. But where is the VPA?

Roo-man:
I think we will take a short break while you study the documentation on Program Pointer Allocation.

A Short Break

CHAPTER 6

Roo-man:
Please assimilate the information you obtained from perusing the allocation document.

Joey:
Do what, from where?

Roo-man:
What did you learn.

Joey:
Oh. Well, the VPA is built directly after the internal endline. The block structure for each type of variable is as follows:

simple numeric variable:	2 byte name form
	2 byte value pointer
string variable:	2 byte name form
	2 byte maximum length
	2 byte value pointer
array variable:	2 byte name form
	2 byte total array length
	2 byte maximum row
	2 byte maximum column
	2 byte value pointer

Roo-man:
You've neglected to mention the VPA entry structure for a user defined function.

Joey:
Well, I sort of forgot to read that section.

Roo-man:
Hmm. Can you answer your original question then?

Joey:
Which one?

Roo-man:

When does the PCB get validated?

Joey:
You mean like parking in front of a fire hydrant?

Roo-man:
I propose we adjourn this discussion for a short time to allow our sponsers time to swallow this bunk.

Later	CHAPTER 8

Joey:
Gee Roo-man, where did you go?

Roo-man:
Out back! Let us return to Pointer Allocation and the PCB. If you haven't already discerned, the final stage of Pointer Allocation is to compute and store the pertinent information, specifically, program length, VPA length and environment length, into the appropriate locations of the PCB.

Joey:
Nope, I wasn't so discerned.

Roo-man:
Sacre Bleu!

Joey:
Can we get back to the program?

Roo-man:
I think this would be a good time to examine the PCB and VPA of our program 'MAGIC'.

The PCB

P.LEN	P.PLEN	P.GLEN	P.ELEN	P.SPAR
92 00	0E 00	00 00	32 00	00 00

The VPA

	name	val ptr		
variable I:	0A 09	1E 00		
	name	max len	val ptr	
variable A$:	8A 01	20 00	26 00	
	name	val ptr		
variable X:	0A 18	48 00		

Roo-man:
When this stage is complete, the process of Pointer Allocation is over.

The next process in our series is termed Environmental Allocation. The environment for a program contains not only the values of the variables but also such information as GOSUB count, FOR/NEXT count, ON ERROR information, Mother program name an...

Joey:
You mean even programs have mothers?

Roo-man:
No, lad. In fact programs do not have mothers. It is the environment which belongs to the mother program. In some cases it is necessary that a program require multiple environments. Consider, for example, a program which uses recursion, oh maybe, a Tchebyshev polynomial generator; this program might require multiple environments during execution. Each of these environments would belong to the same program i.e. it's mother.

Joey:
How do you find the environment? Who builds it? Is it always there? How big is it? Where is the programs mother? Can I go play hoop now?

Roo-man:
HOOP?, whats hoop?

Joey:
Hoop is hoop, b-ball, you know, Dr.J's demain.

Roo-man:
Dr. J?

Joey:
Are you serious? You don't know who Dr. J is? Well! I think we should take a short break untill you can demonstrate to me just who Dr. J is!

Roo-man:
Get back here!

Roo-man's Impressions of Dr. J	CHAPTER 9

Next Day	CHAPTER 10

Joey:
Good morning Roo-man.

Roo-man:
That is strictly a matter of opinion. Back to work please. What do you remember about a programs environment.

Programer:
Let's see. That is where the varialble values are stored and neat information about the program is also stored there. Oh, yea, the programs mother is there if the program was CALLed.

Roo-man:
One point of clearification. The mother program does not reside in the environment, simply the NAME of the mother program. Certainly, one may visualize the structure of an environment. For example, given the constraints of our machine how many bytes are necessary to store the value of a simple numeric variable?

Joey:
It depends.

Roo-man:
Very good, but depends on what?

Joey:
Whether the variable was declared a REAL, INTEGER or SHORT. If the variable is a REAL then 8 bytes are allocated as a value area, an INTEGER is allocated 3 bytes and a SHORT is allocated 4 bytes.

Roo-man:
My young man, that is commendable! May I go so far as to inquire the structure for a string variable.

Joey:
No you may not.

Roo-man:
I asked for that. A string variable may be dimemsioned by declaration in a DIM statement. This is not a necessity but if the Joey knows the limits to which the variable will be utilized, declaration of the string length will optimize the use of memory. The only restriction on the DIM statement is

that it must contain the very first reference of the string variable in the program. Similarily, arrays should also be dimensioned.

Joey:
What happens if I use string and array variables but don't declare them in a dimension statement? Will I get an error?

Roo-man:
Ah the beauty of default values. No, Kangaroo is a very lenient machine. If a string variable is not declared, it is allocated 34 bytes of room in the environment. The string value field is comprised of two parts. The first 2 bytes of the string value field are reserved for the strings actual length and, in fact, declared or not, the first two bytes of any string value field are reserved specifically for this purpose. In the default case, the remaining 32 bytes contain the sequence of characters that comprise the string variables value.

Joey:
Oh, so the max length portion of the VPA entry for a default length string will have a value of 34? Or will it have a value of 32?

Roo-man:
What is the maximum number of characters allocated for a default string?

Joey:
32 bytes.

Roo-man:
You have just answered your own question. Lets continue. As I have stated, Kangaroo will also assign default dimensions to an array variable if not declared. Here, however, there is potential for confusion. Have you ever utilized the OPTION BASE command?

Joey:
Yea. Sometimes it is easier to have the first element of my array to be referenced as the zeroth element. Other times it is easier to begin the array with element 1.

Roo-man:
Because of the possible ambiguities that could arise, the OPTION BASE command must appear before any reference to an array variable. Although this sets the initial subscript value for the array, it in no way affects the default value or values for the arrays dimensions. The maximum row and if necessary the maximum column default to 10. Also the option base defaults to zero. Suppose I referenced the variable A(i,j) in the first line of a program; Please describe the values of the VPA entry for this variable. You may exclude the value pointer.

Programer:
Well, the name form will have information in it that says that the variable is an array, its name is A and it's elements are REAL numbers. The maximum

row and maximum column will be 10. Lets see, 11 times 11 is 121 and 121 times 8 is ah... 968. So in hexidecimal the max row and max col would have 00 0A and the total length would have 00 68.

Roo-man:
Very close actually. Here is a scenerio. Pointer and Environmental allocation have concluded. We are now entering execution coce and we attempting to display the value of the array element A(2,3). The array A(,) was not declared in a DIM statement. Find the element A(2,3).

Joey:
Okay. We find the beginning of the array from the value pointer. Is the option base 0 or 1?

Roo-man:
Exactly!

Joey:
Huh? Oh is see. Somewhere in the VPA entry we should tell whether the option base is 0 or 1.

Roo-man:
There is hope for you yet. Specifically, the obvious place is in the total length field. The most significant bit of the total length field is set if the option base is 0; other wise the option base is 1. So, back to the VPA entry of our original array A(i,j). What should the total length field have been?

Joey:
It should have been 80 08. Right?

Roo-man:
Correct. Given this background, let us discuss the creation of the environment. Recall the PCB after Pointer Allocation. Notice that length of the environment is already determined. It is this value plus the length of the information field that is allocated for the programs environment. The information field is called the Environmental Control Block or ECB and its length is 30 bytes. Please regard its structure.

bytes	name	description
0/1	E.LEN	length of environment including the ECB
2/3	E.PREV	length of previous environment
4/5	E.RMEM	amount of memory reserved by program
6	E.FCNT	FOR/NEXT count
7	E.GCNT	GOSUB count
8/9	E.EREX	address of code to exec upon an ON ERROR
10/11	E.ERPC	Kangaroo PCR after ON ERROR
12/13	E.ROM	ROM number of mother program
14/21	E.MOM	name of mother program
22/23	E.RTN	R10 for CONT/RUN

24/25	E.PCR	PCR for CONT/RUN
26	E.STAT	current status R16 (xstat)
27	E.DATA	location in current DATA line
28/29	E.DATL	pointer to current DATA line

The ECB is always the first 30 bytes of a programs environment. It is the responsibility of Environmental Allocation to initialize the values of the ECB and the variable value area. Do you recall what happens when you access the value of a variable that has yet to be defined?

Joey:
Yea. You get a warning 'no value' and if the variable was a numeric a 0 is returned or if the variable was a string the null string is returned.

Roo-man:
Yes. Any ideas on how the machine can tell a no value situation versus a zero or null existence?

Joey:
Well, the variables value area could have a no value situation flagged.

Roo-man:
Are you the same person who was here yesterday? That is very astute. In the numeric variable case, the no value situation is flagged by setting the most significant byte of the field to FF hexidecimal. Whereas, a string variables value is not defined if the two byte length field is FF FF hexidecimal. Thus, the final responsibility of Environmental Allocation is to initialize all value fields to the undefined state. Shall we examine the environment for 'MAGIC'?

ECB:	31	00	1F	00	00	00	00	00
	00	00	00	00	00	00	40	41
	47	49	43	20	20	20	20	00
	1C	00	02	00	00	00		
var I:	00	00	00	00	00	00	00	FF
var A$:	FF	FF	20	20	20	20	20	20
	20	20	20	20	20	20	20	20
	20	20	20	20	20	20	20	20
	20	20	20	20	20	20	20	20
	20	20						
var X:	00	00	00	00	00	00	00	FF

As you can see, the ECB contains valid information about the program 'MAGIC' and the three variable value fields have been initialized. We are ready for execution!

Part 1 Synopsis	CHAPTER 11

Narrator:
A quick recapitulation:

Once a programmer has logically charted the flow of his/her program, the time for creation has come.
As each program line is entered into the Kangaroo display and the return key is pressed the process known as Parsing is initiated. The system Parser interprets each keyword into internal tokenized form and structures the tokens into RPN form. Finally, the appropriate memory location is found and this stream of tokens is inserted at this location.
The next phase, Pointer Allocation, is then begun. Pointer Allocation builds a VPA, computes the size of the environment and replaces variables and line numbers with relative pointers.
The last stage before execution is allocation of the programs environment. The ECB is constructed and the variable value areas are initialized to the undefined state. We now return to the regularly scheduled dialogue.

Roo-man, Redux	CHAPTER 12

Roo-man:
Well Joey, ready for another session? Fine. We turn our attention to the execution phase of our program 'MAGIC'. The controller of execution is the Interpreter loop. It is the Interpreter's responsibility to 'interpret' each token in our program into an absolute address at which resides the respective runtime code. After each token is executed the interpreter checks the STALL bit (the msb of R17) to decide whether to continue or not. If this bit is set them the interpreter loop is halted.

Joey:
Why would this bit be set?

Roo-man:
There are three reasons for a program to halt before the END statement is executed. The first is the execution of the STOP statement. The programmer may decide that a STOP command is necessary in the program. When the interpreter process the STOP token the program is halted and the state of the programs environment is preserved. This means, the programmer may inspect the value of any variable used in the program and compare this to the expected value.

Joey:
Golly, Roo-man. That could be useful in debugging a program, huh?

Roo-man:
Very useful. A second means of stalling a program is to hit the ATTeNtion key. This action also halts the program by setting the msb of R17. Note the difference between this and halting by STOP. The STOP token is totally devoted to stopping a program at a particular location, whereas hitting the ATTeNtion will halt the program at what might appear to be an arbitrary location, not necessarily a usable location to the programmer. The third means of stalling a program requires no propgrammer interaction. A program is stalled if a runtime error occurs.

Joey:
What is a runtime error?

Roo-man:
A runtime error is any error which occurs during the execution of tokens. There are also Allocation errors and sometimes these are hard to distinguish from runtime errors. The biggest factor is the appearence of a line number in the error message. If a line number appears then the error

is most likely a runtime error, if not then the error is most likely an allocation error. These are the three ways to stall a program. In each case, the programs environment is still valid (as valid as can be). Let's replace the END statement in line 80 with a STOP statement and look at the environment after execution has been halted.

	51	00	1F	00	00	00	00	00	00	00
ECB:	00	00	00	00	4D	41	47	49	43	20
	20	20	58	00	54	00	02	00	00	00
var I:	20	20	20	20	FF	11	00	00		
var A$:	07	08	66	72	6F	67	67	69	65	20
	20	20	20	20	20	20	20	20	20	20
	20	20	20	20	20	20	20	20	20	20
	20	20	20	20						
var X:	00	08	59	53	26	59	41	31		

Notice the variable values. Since I is an integer the first four bytes are not valid information. The fifth byte FF hex is a flag to the system that this is an integer value. The remain three bytes are the value of I, in this case I=11. Variable A$ has an actual length of seven bytes and is equal to 'froggie'. The variable X is a real value and is equal to 3.14159265359. Any questions?

Joey:
Is the programs environment valid all the time?

Roo-man:
Afraid not Joey. When the interpreter process the END token, the programs environment is removed in order to free up memory. However, the PCB and VPA are still valid so that the next execution of the program will proceed that much quicker.

Joey:
What happens if I edit the file? Won't that cause trouble for the PCB and VPA? I mean, what if I add new variables to the program? There son't be any VPA entry for them.

Roo-man:
Ah so, we arrive at the final chapter

Final Chapter	CHAPTER 13

As I was saying, we have arrived at the final process in the life cycle of our program. From a fresh and virile sequence of logical statements to a warped and feeble menagerie of lost thougths.

Joey:
Heaavveeyy!

Roo-man:
Thank you. We speak, of course, of the process of DEALLOCATION!

Joey:
You mean we're going to undo all we've done?

Roo-man:
That's right! When the programmer EDITS, COPIES , MERGES, PURGES and all sorts of other things to his program, the program must be deallocated. The PCB is wasted! The VPA is disintigrated! The variable pointers are ambushed by their godless variable names! Furthermore, if there are any external ROM defined tokens and their ROM is not available, the WHOLE LINE in which that token appears is blown away and replaced with 'ROM missing'!

The Absolute Final Chapter!	CHAPTER 14

Roo-man:
Well Joey. That's my story for this week. Did you enjoy it?

Joey:
Yep

Roo-man:
That's not how you spell Yap!

Joey:
Yes it is! Just go talk to the guy who is always putting words in our mouths!