

RA™

"A small yet powerful language for the Macintosh™."

USER DOCUMENTATION

(Preliminary Release)

**By Serg Koren
(SKOREN - GENIE)
(70037,1405 - CIS)**

**VER 1.0
10/18/86**

©1986 by COM~LINK and S.Koren.
All rights reserved.

Macintosh™ is a trademark of Apple Computer.
RA™ is a trademark of COM~LINK.

TABLE OF CONTENTS

2	TABLE OF CONTENTS
3	INTRODUCTION
4	SHAREWARE NOTICE & BENEFITS
5	THE RA SYSTEM
6	RA -- THE PROGRAM
9	THE MENUS
9	Apple Menu
9	About RA™...
14	File Menu
14	New
15	Open
16	Save
16	Save As...
16	Revert
17	Read RA/Read Text
17	Quit
18	Edit Menu
19	Options Menu
19	Run
19	Step
20	Instruction Select
21	Windows Menu
21	Stack
22	Environment
23	Hide Input/Show Input
23	Base
23	String Stack
24	RA -- THE LANGUAGE
24	BASICS
26	MATH OPERATORS
27	BOOLEAN (LOGICAL) OPERATORS
28	CONDITIONALS
29	LOOPS
30	VARIABLES
31	SUBROUTINES
33	STRINGS
34	I/O
35	PROGRAMATIC CONTROL OF OUTPUT WINDOW
36	MACINTOSH TOOLBOX CALLS
39	MISCELLANEOUS COMMANDS
40	THINGS TO COME
41	THANKS TO..

WARNING & DISCLAIMER: Use RA on its own disk, and always use a backup copy!!! We have attempted to make RA as safe to use as possible. But because it is a programming language and because it has certain low-level commands involving memory and the TOOLBOX, we cannot control what you program and create using RA. Therefore, we will not be held liable for any vital data you may lose if you do not read this disclaimer or decide to put RA on the same disk as your bosses '87 fiscal budget disk, or what-not. However if you are registered, and do cream your master (use backups) we will send you the latest copy of RA if you send us a blank disk.

INTRODUCTION:

RA -- The Egyptian sun god. The god of right over wrong, light over darkness, truth over falsehood. He was synonymous with movement. Ra was usually depicted in human form; sometimes with and sometimes without the head of a hawk.

Welcome to RA. RA is a programming language written for the Macintosh computer. RA was designed to be simple to learn yet powerful enough to be useful. That is, the syntax of RA is very straight-forward; yet RA has facilities for handling the Mac toolbox, Macintosh (not in Ver.0.1), etc. In its current incarnation RA is an interpreted language. In the future it will be both interpreted and compiled. Currently, RA deals with integer arithmetic only; in future incarnations it will handle the SANE package floating point system. RA also has the ability to help debug the code you write as well as provide a program editing/development environment from within itself (you don't need EDIT/Switcher, etc -- unless you want it.) Sample programs are also provided.

RA will continue to grow in the future and the direction of that growth will be in part determined by you the user. Please feel free to report any problems, bugs, suggestions, and/or comments that you may have. We will attempt to resolve any of these as quickly as possible. The only thing we ask is that if you are having problems with a piece of code you have written using RA, you include a copy of that code in your bug reports. This is to allow us to see exactly what you are referring to.

A PLEA

RA is not free. It is shareware. We are asking \$10.00 for this version of RA. If you do not like what you see or can't use it, pass it on to someone else, or erase your copy (if you can't use it, why keep it?) Better yet, write to us and tell us how to make RA better so that it can better serve your needs. And if your points are valid/interesting and have enough of them (we are the final judge of what is) we will waive your \$10.00 or refund it. Can't get much fairer.

What do you get for your \$10.00? You become registered with us and any problems/questions you may have will be given priority over those who don't register. You will also be notified by mail of future releases (the compiler, etc.). In addition, you get in on the 'ground floor'. That is, the shareware fee for future versions will probably be \$30.00. As a registered owner, you will receive the next three incarnations free, after which point they will be \$1.00 for each upgrade. For those of you thinking of holding out and paying the \$30.00, the upgrade will be \$3.00 for each after the first three free.

We've tried to be fair about the shareware policy. The only thing we ask in return is that if you decide not to register, please get rid of your copies in one way or another. Thanks.

Another incentive: If you register, and come up with a nice little program written in RA and you send it to us we'll extend your free upgrades from the number you have currently left by one. If your program in our estimation is worth it we'll send you a disk full of RA programs and samples FREE!

Thanks for registering and your support!

THE RA SYSTEM

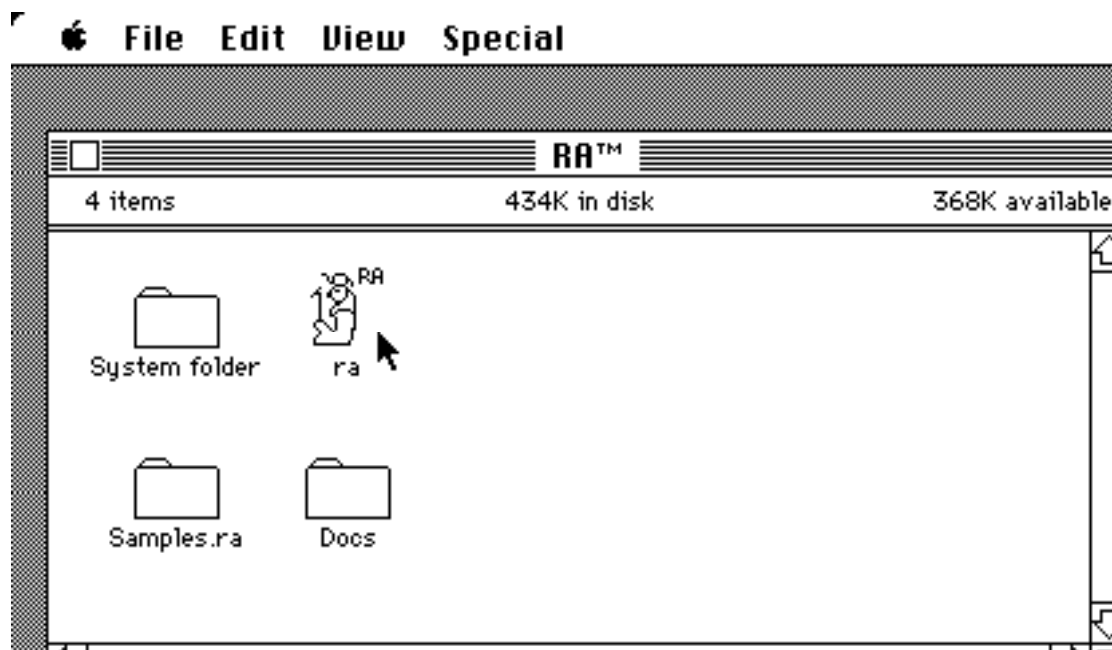
You should have on your system disk:

RA	The actual interpreter/compiler/editor.
RA Samples	A folder full of RA executable source files.
RA User Doc	This documentation file
RA Bugs	A text file containing all known bugs.
RA Bugs ACTA	An ACTA outline file containing all known bugs. You need ACTA to use this version.
RA Bugs Report	A MacWrite 4.5 file to be used to submit a bug report via mail.
RA Registration	A MacWrite 4.5 file to be used to register your copy of RA with us.

NOTE: Ver 0.1 does not include the RA-Bugs and RA-Bugs-ACTA files.

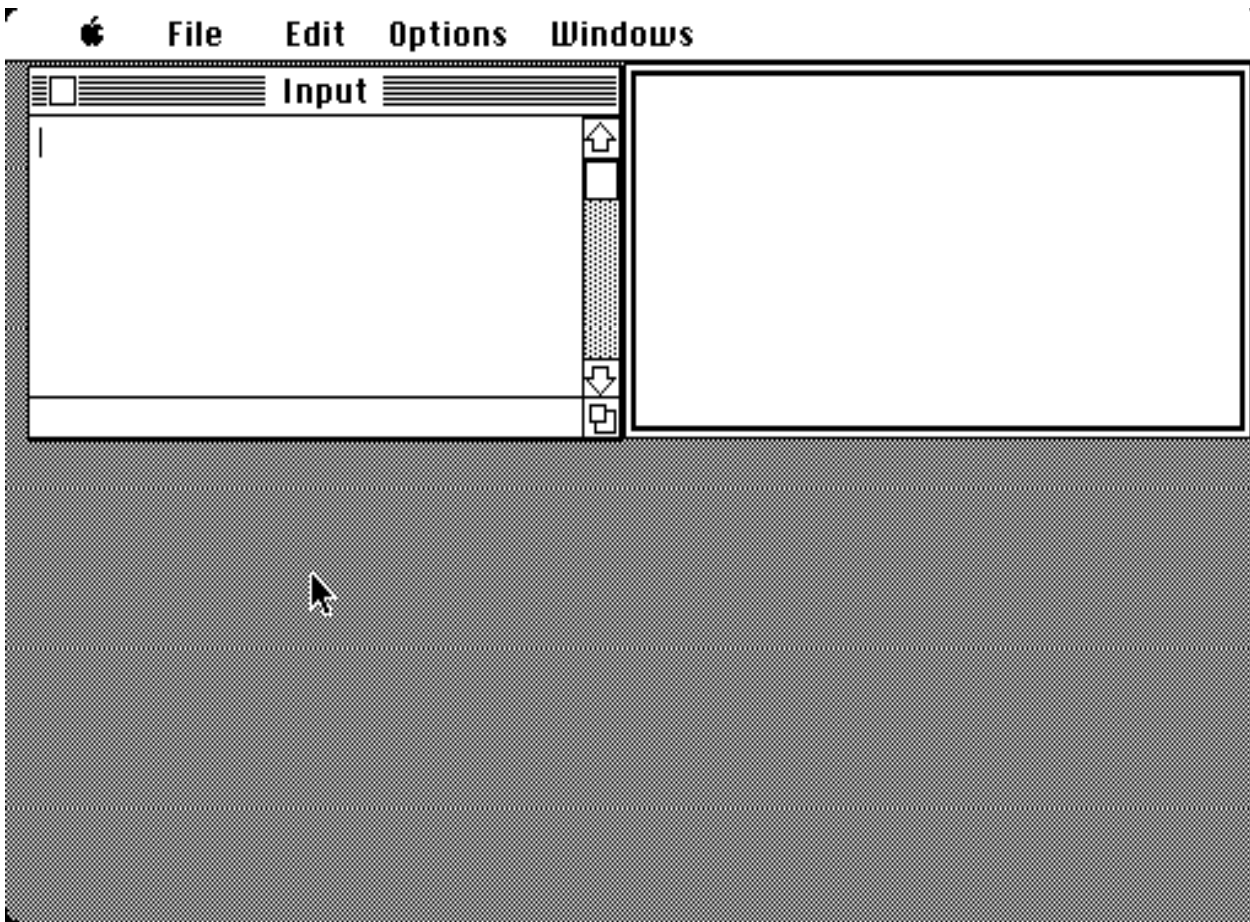
RA -- THE PROGRAM

To start, find the RA icon on your desk:



and double-click on it or open it from the 'File' menu.

The RA system will be entered and the following should appear on your screen:



The only windows which should be visible on the desktop are the 'Input' window and an 'Output'. The 'Input' window is a text-edit window which supports all the standard ways of editing and creating text. This window can be resized, dragged, etc. The 'Output' window is the one without a drag/size/close region. This window is used to display output to the screen. This window can be controlled from within the RA language so that it may be resized programatically.

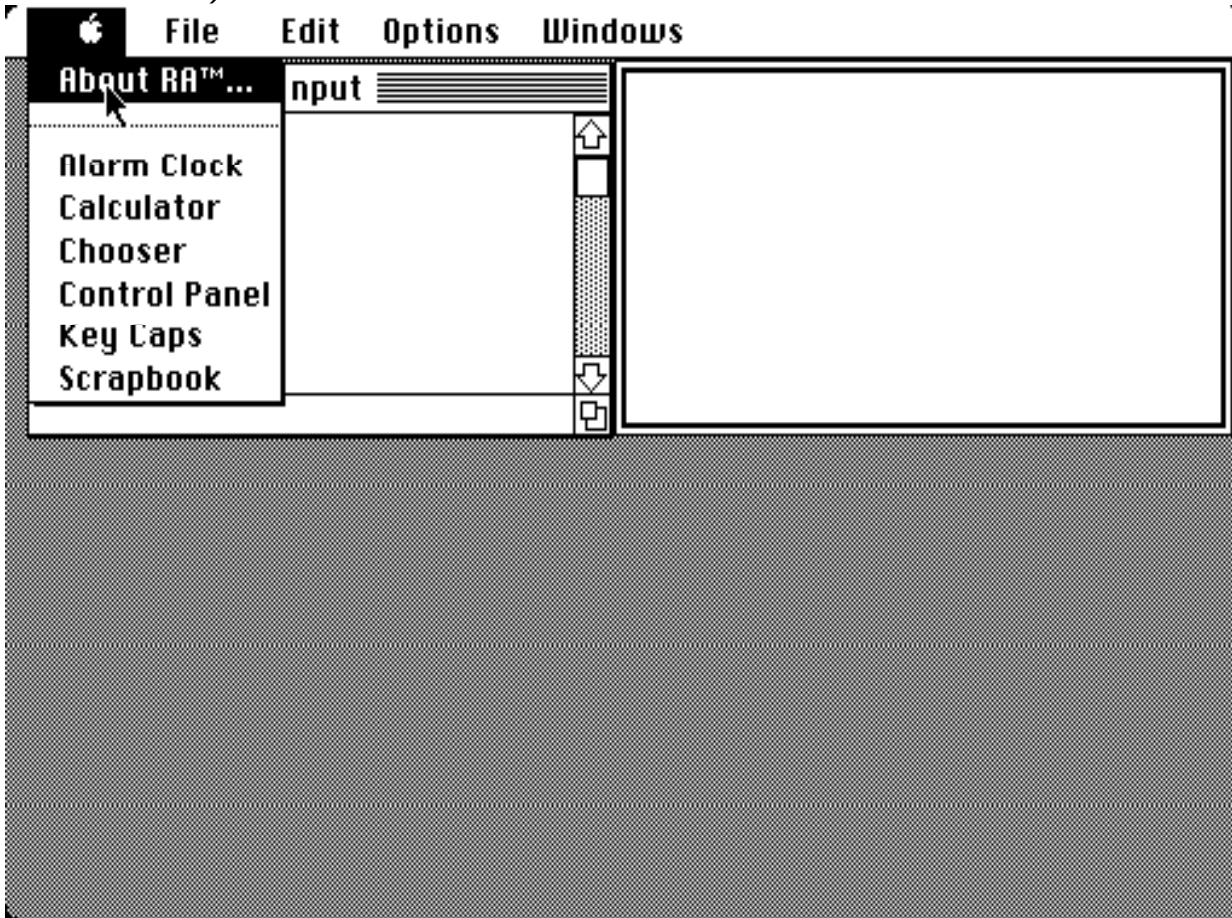
Other windows may become visible depending upon what you are doing.

THE MENUS

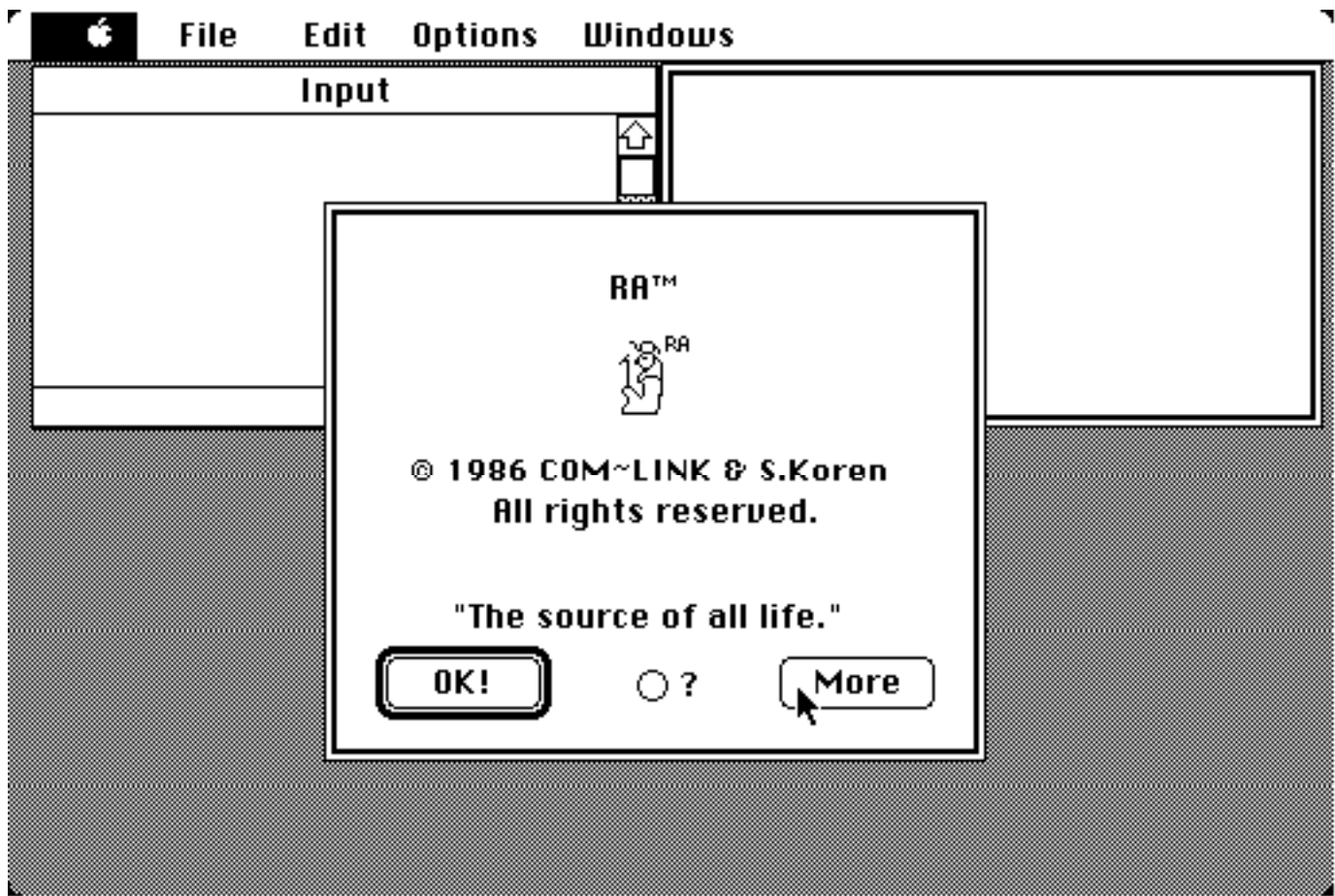
1 - Apple

This menu allows you to access any installed desk accessories and lets you get information about RA™.

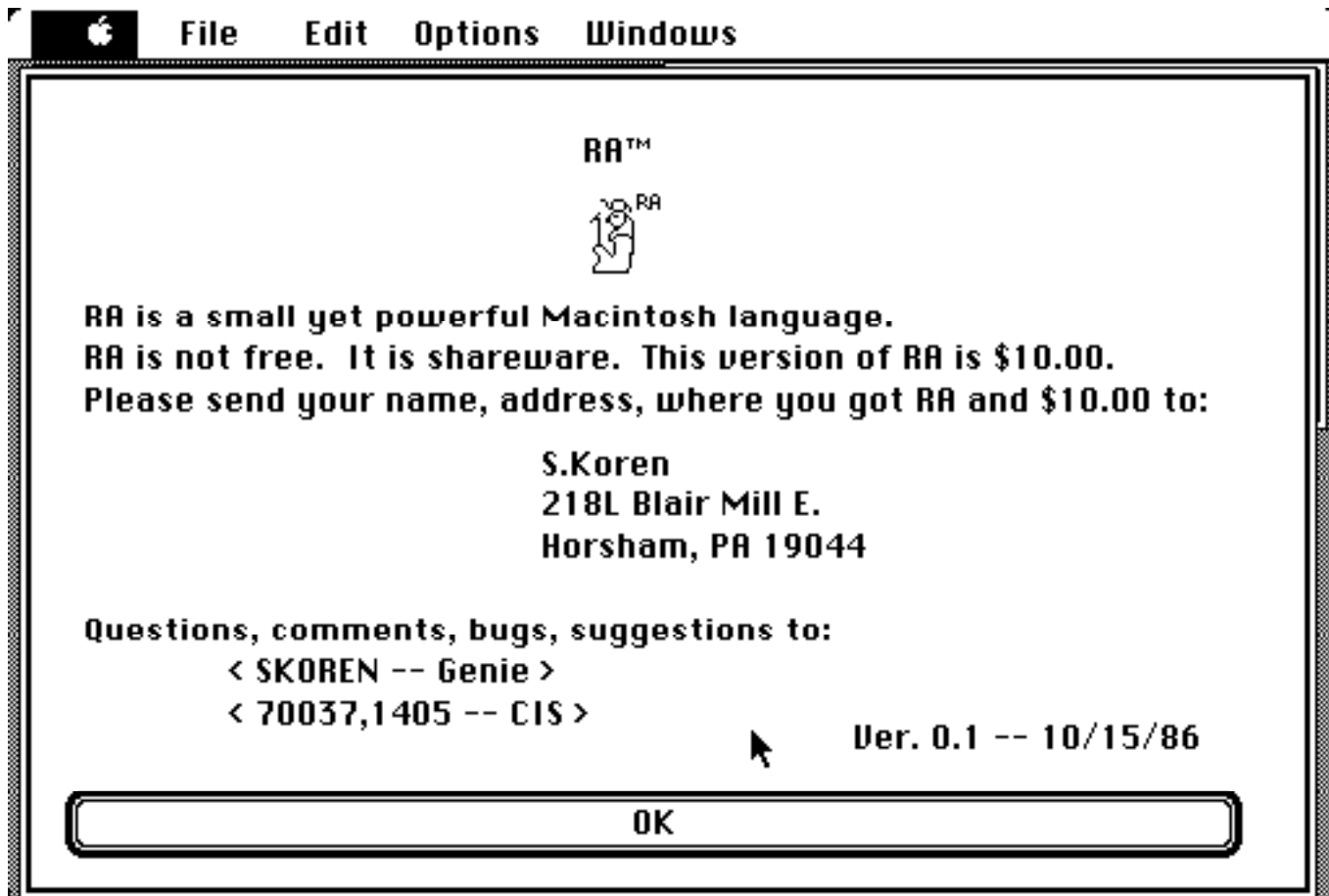
a) *About RA™...*



Clicking on this item will bring up the 'about' box which gives the name of the program.

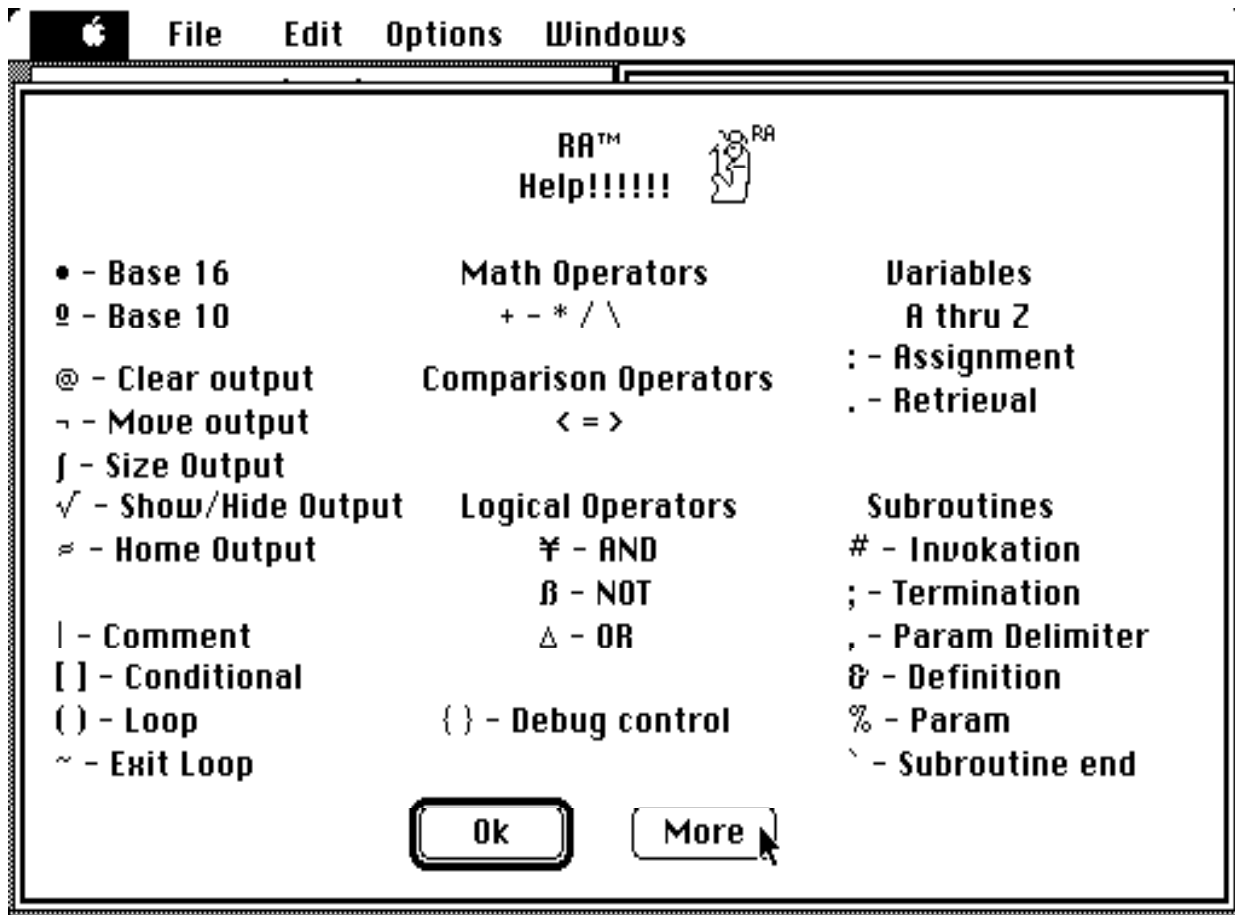


The 'More' button of this dialog provides shareware and version information.

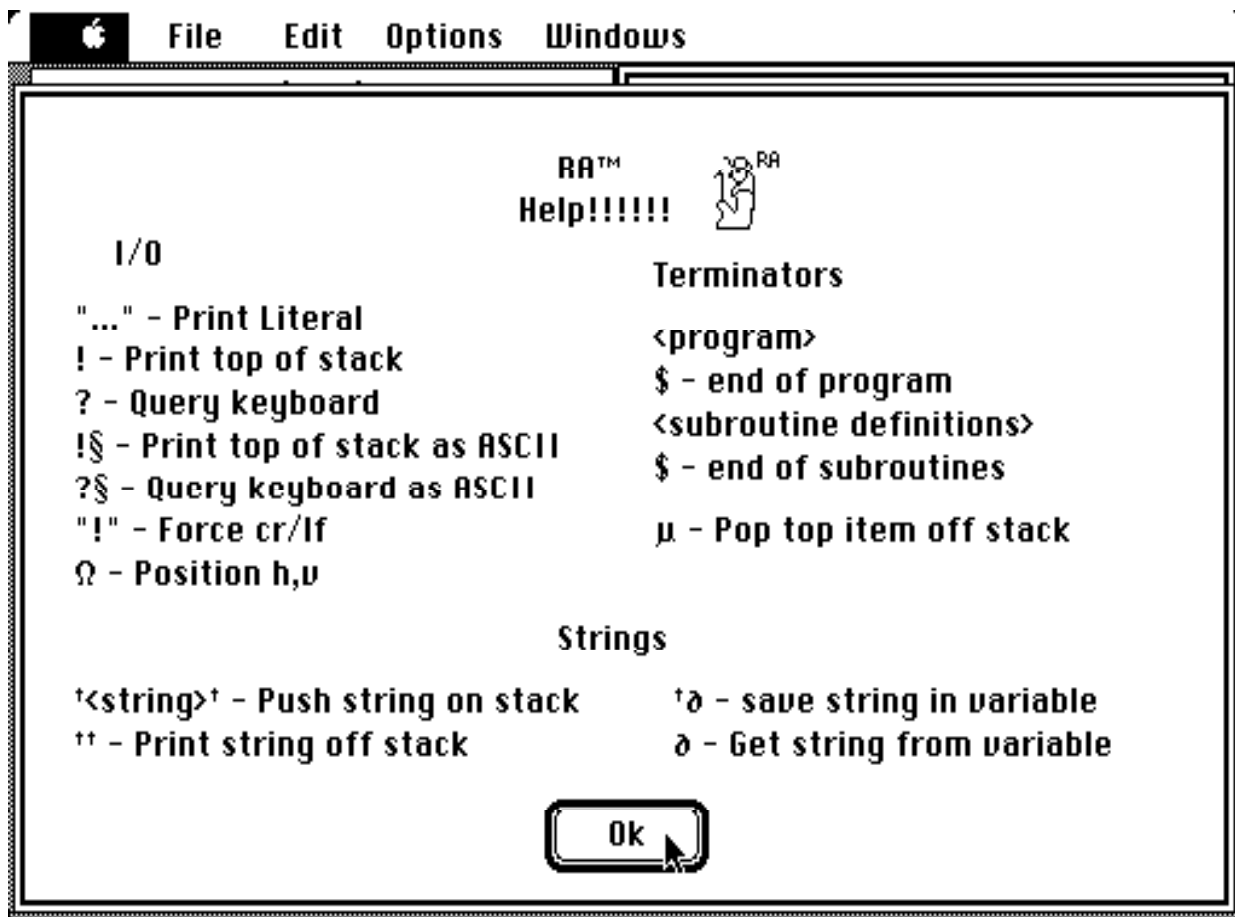


Clicking on the 'Ok' button of the shareware dialog will take you out of the 'About' item.

The '?' button of the 'About' Dialog is used to obtain help information about RA™. Clicking on the '?' button will display the following screen:



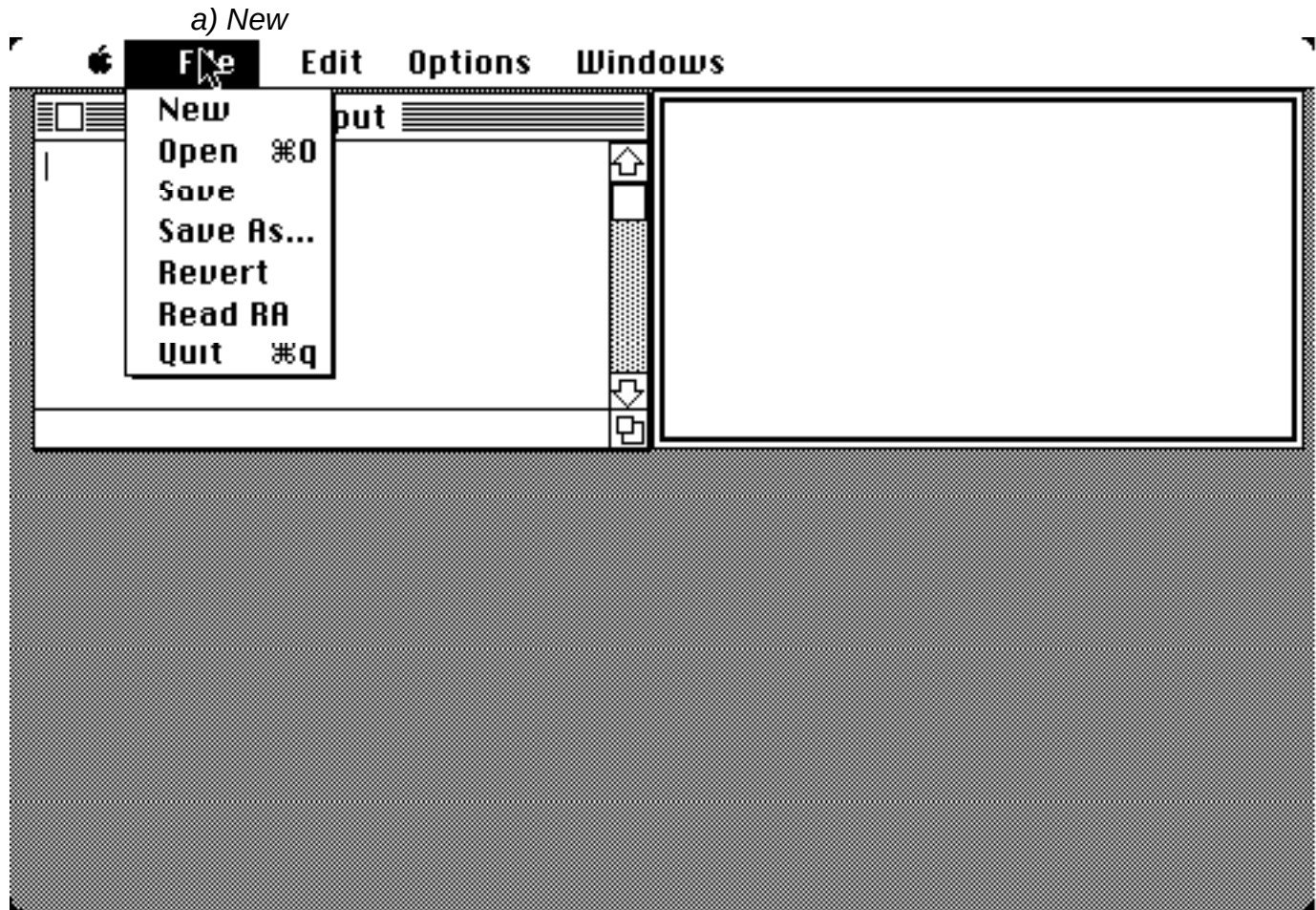
Clicking on the 'Ok' button will take you out of the 'About' item. Clicking on 'More' will display the second help page:



Clicking on the 'Ok' button of the second page of the Help information will cause you to leave the 'About RA™...' selection.

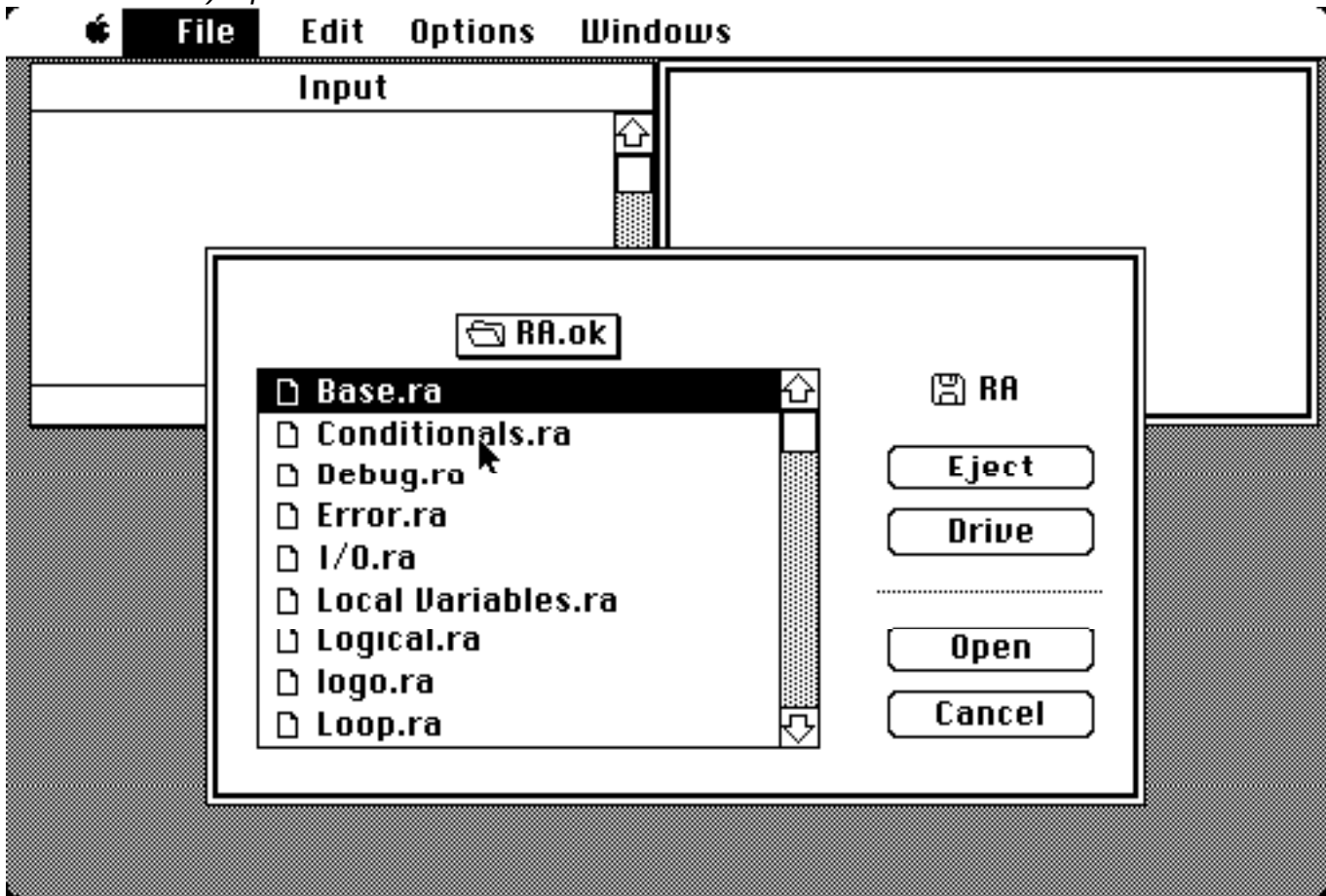
2- File

This menu allows you to perform file related operations and contains most of the standard Macintosh 'File' menu items.



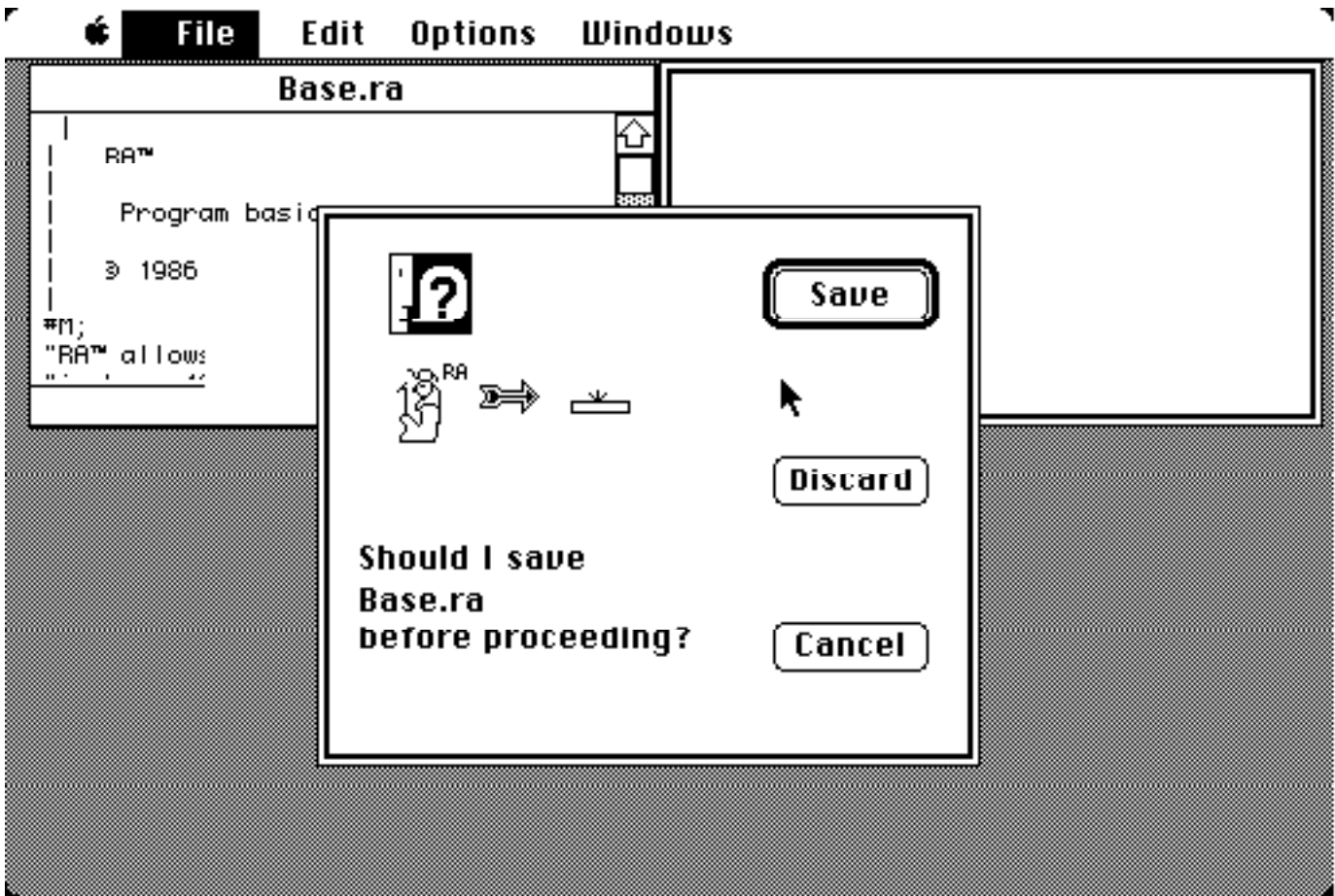
Selecting the 'New' menu item will create a new input window. If the previous input window had information which was not saved (i.e. an open file) RA™ will display a dialog which will prompt you for confirmation, or cancellation. (This dialog will be explained later.) The newly created window will not be associated with any file and will be completely empty.

b) Open



Choosing the 'Open' item from the 'File' menu will bring up the standard SFGGetFile dialog. You may then select which file will be opened and read into the 'Input' window. The files which are selectable from the 'Open' item are dependent upon the state of the 'Read RA/Read Text' item of the 'File' menu.

If the 'Input' window has information which has not been saved to disk, the following dialog item will be displayed:



This dialog will be displayed on both a 'New' and 'Open' selection. Hitting the 'Save' button will save the current contents of the 'Input' window to disk, hitting the 'Discard' window will open the selected file, or create a new 'Input' window without saving to disk, and the 'Cancel' button will abort the 'New' or 'Open' event.

c) Save

Selecting this item will save the contents of the 'Input' window to the currently opened file name.

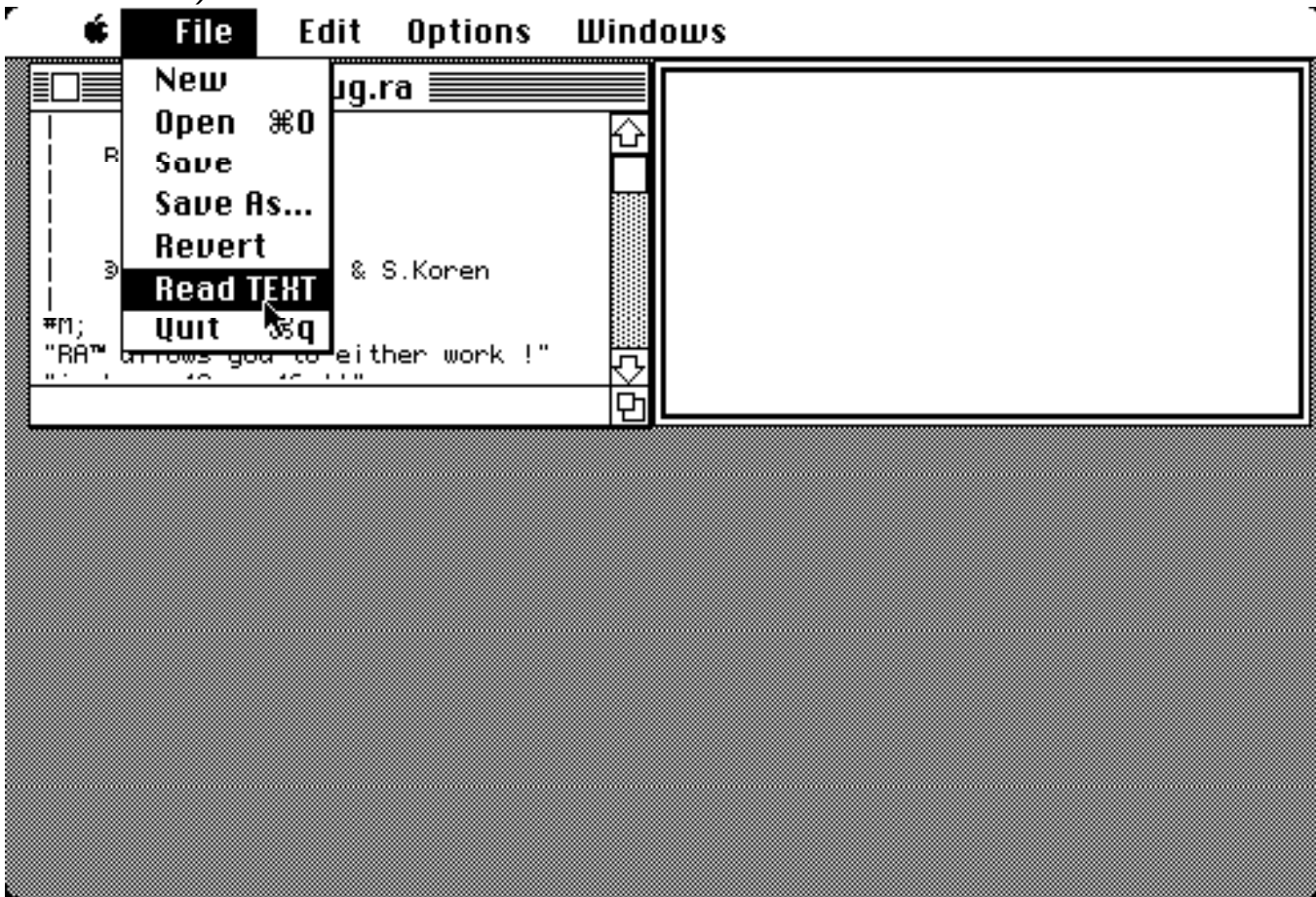
d) Save As...

Selecting this item will save the contents of the 'Input' window to a user-chosen file. A standard SFPutFile dialog will be displayed, for the user's use.

e) Revert

This item will cause the last saved contents of the 'Input' window to be read into the 'Input' window. A dialog will be displayed prompting the user for either confirmation or cancellation.

f) Read RA/Read Text



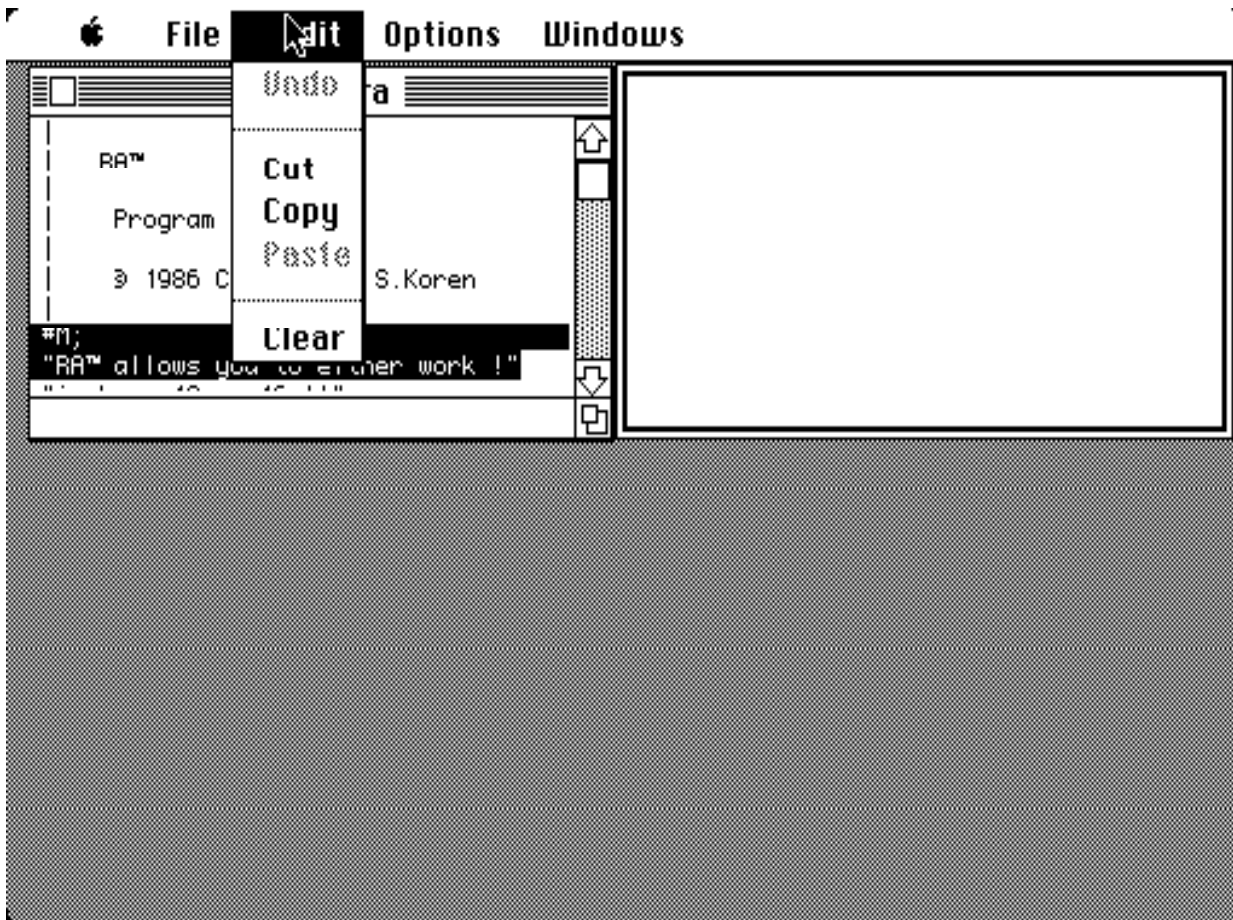
Selecting this item will cause it to toggle between one of two settings: 'Read RA' and 'Read Text'. The current state of the item indicates what type of file is to be read into the 'Input' window. That is, 'Read RA' will read RA-created files, 'Read Text' will read files created by other editors (as long as the file type is TEXT.) Selecting this item will cause the 'Open' item to filter the appropriate file type to the user.

g) Quit

Selection of this item will cause RA to close any open files (after prompting the user), and return the user to the Finder.

3) Edit

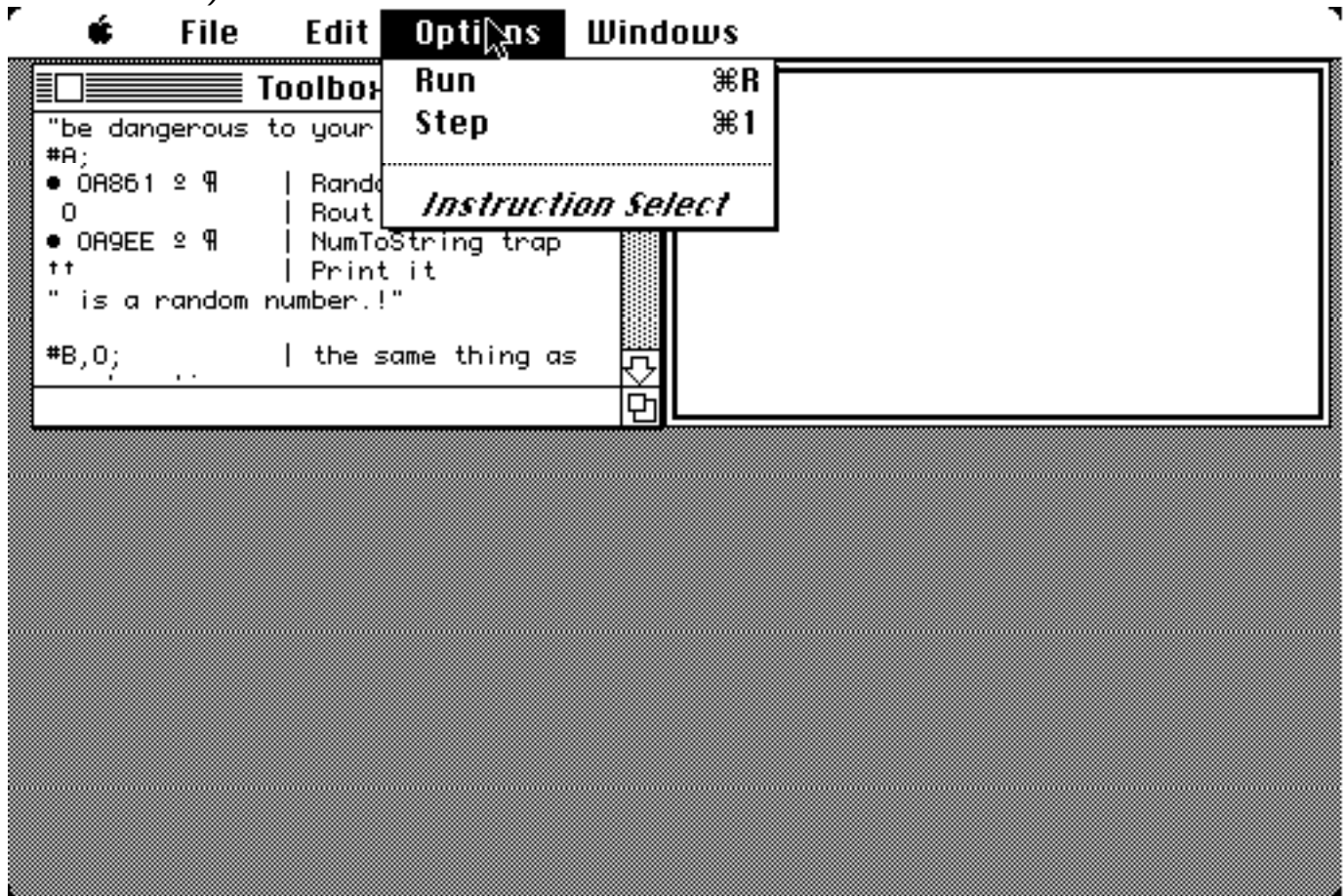
This menu is the standard Macintosh Edit menu. and functions in a standard manner. However, the 'Undo' item is not implemented and will beep if selected.



4) Options

The Options menu allows you to control various ways of the running programs. Currently, the only options available are those which deal with the interpreter.

a) Run

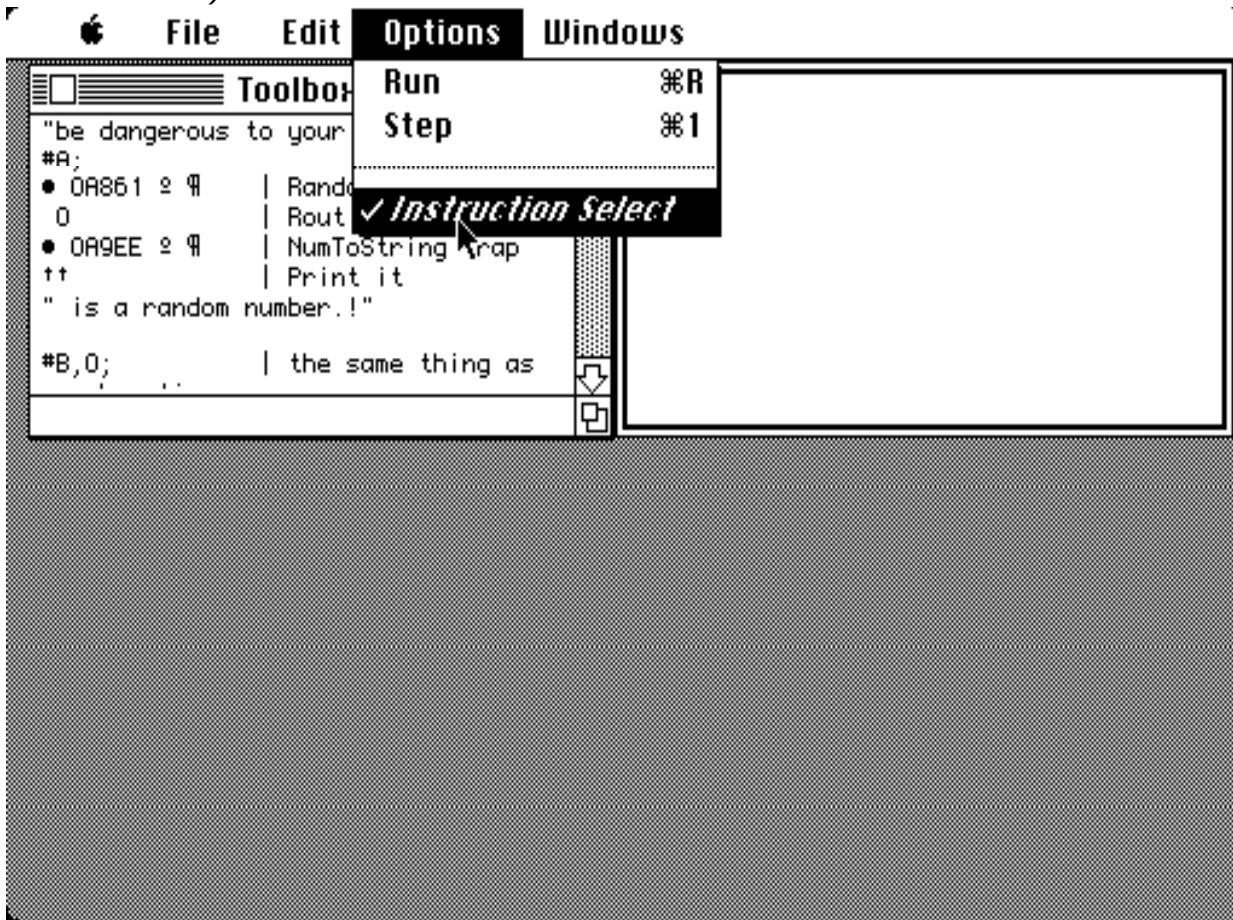


Selecting 'Run' from the 'Options' menu begins execution of the RA™ code located in the 'Input' window. The 'Run' command is an interpret command.

b) Step

The 'Step' item of the 'Options' Menu causes the interpreter to execute one step, or token, of the RA™ language. If the end of the program is reached, the 'Step' command will begin executing the RA™ code from the beginning of the file. This item is most useful in debugging code when the next item is selected.

c) Instruction Select



The 'Instruction Select' item of the 'Options' Menu is a toggle item. It can be toggled on or off, and its current state is represented by the presence or absence of the check-mark on the item. If a check-mark exists, the item is selected, otherwise it is not.

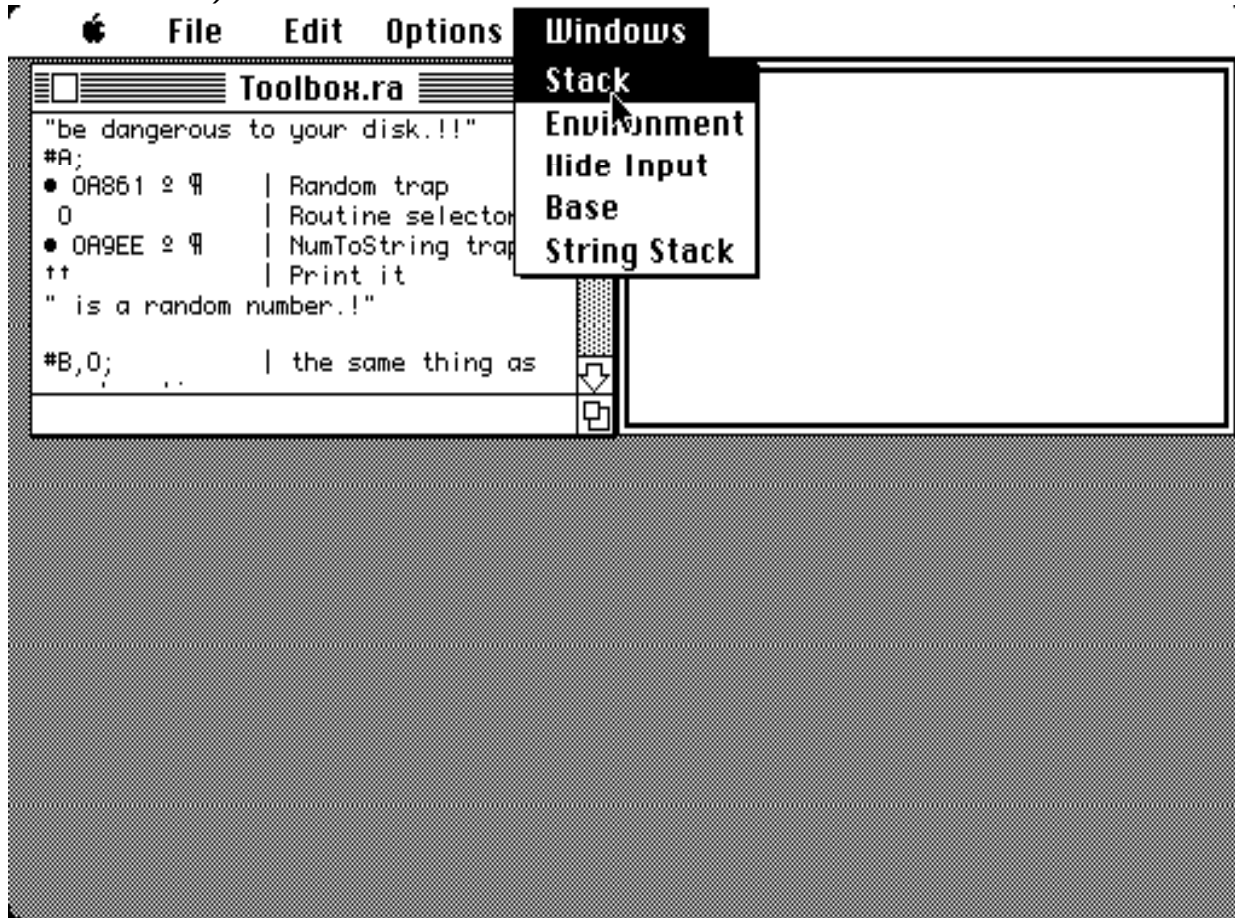
Selecting 'Instruction Select' causes each instruction executed by the interpreted (via 'Run' or 'Step') to be 'selected' or highlighted in the 'Input' window. Choosing this item allows you to see exactly what instruction is being executed, and in this way allows you to debug your programs.

NOTE: This feature can also be enabled/disabled from within a program. This allows you to turn on 'Instruction Select' in a particular section of code and turn it off after the section of code in question is passed.

5) Windows

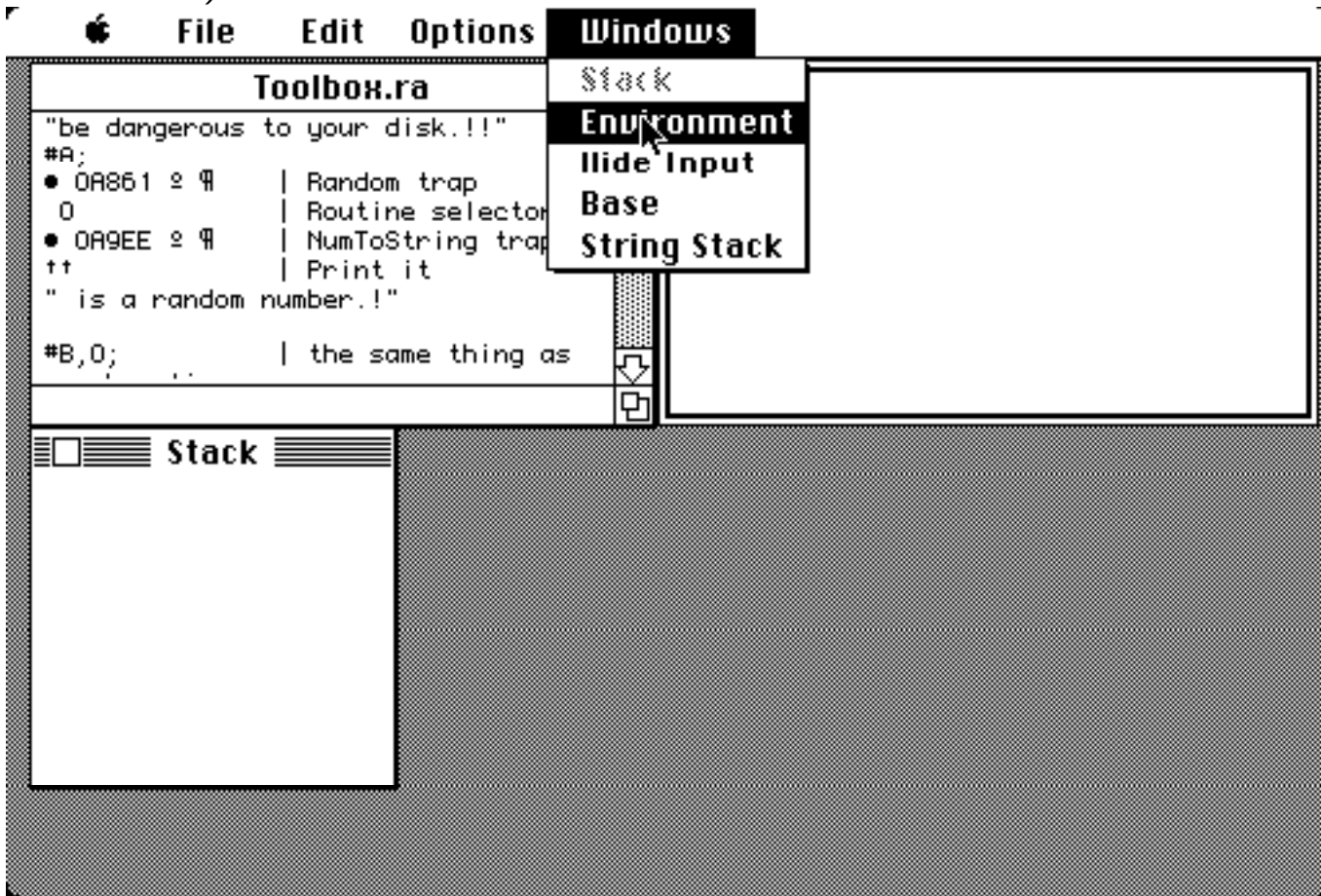
The Windows Menu allows you to display various pieces of information about the current state of the interpreter and the data it is dealing with.

a) Stack



Selecting the 'Stack' item causes a window, which displays a portion of the number stack, to appear. The stack will grow and shrink from the top of the window downward. An empty window implies an empty stack.

b) Environment



The 'Environment' item will display the Environment Stack window on the screen (see the next illustration.) This stack grows and shrinks in the same way as the number stack. Currently, the Environment Stack contains three parts:

ENVIRON

Subroutine - A subroutine call encountered

Loop - A loop encountered

Param - A subroutine parameter encountered.

PC

(the program counter to return to)

DATA LEVEL

(the level at which variables are stored/retrieved.

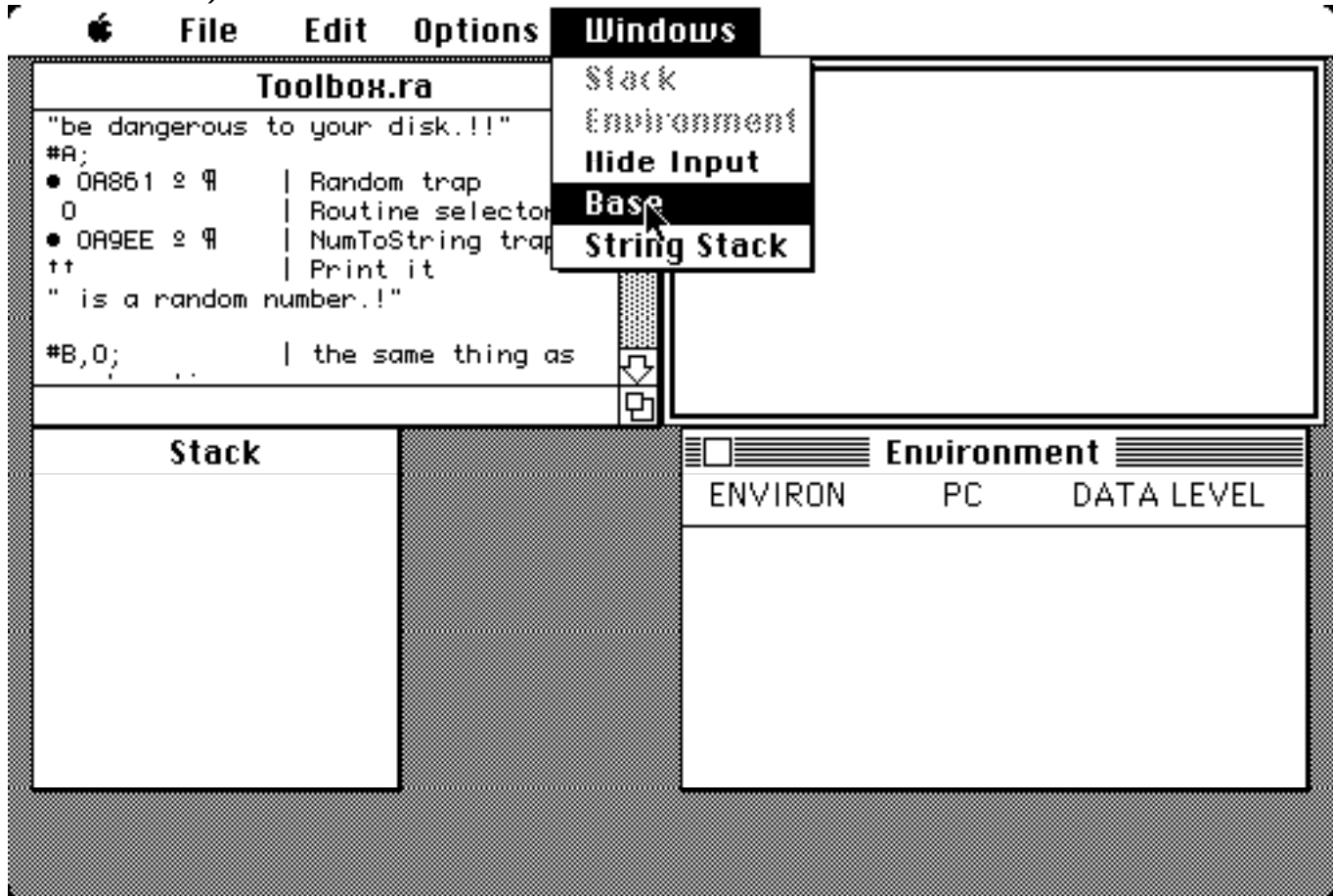
A level of 0 implies global, 26 is the first nesting,

52 is the second, etc.)

c) Hide Input/Show Input

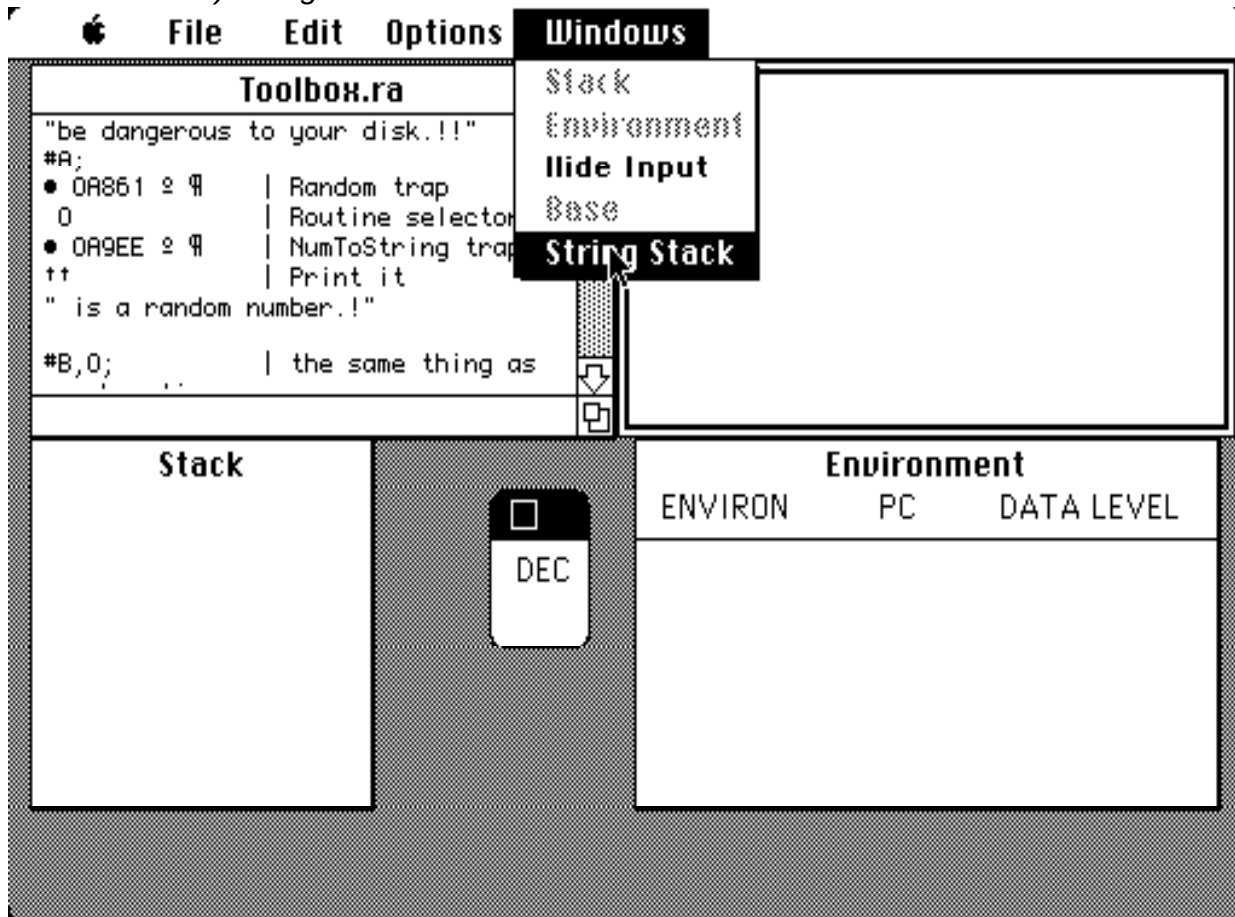
This is an item which will either hide or show the input window depending upon the current state of the 'Input' window.

d) Base



The 'Base' item displays a small window whose only contents is either 'DEC' or 'HEX'. The contents indicates the current base all input which RA™ encounters will be interpreted as. DECimal or HEX.

e) String Stack



The 'String Stack' item displays another stack window. The contents of this stack are all strings used in the RA™ program. This stack will shrink and grow in the same way as the other stacks.

RA -- THE LANGUAGE

BASICS --

RA has been designed to be simple to use and learn. RA is similar to many languages and calculators which use a notation (way of programming) known as Reverse-Polish. In math class we all learned to add two numbers by writing:

$$1 + 5$$

In Reverse-Polish you need to be more logical. You must first tell what numbers the program is to use

$$1 \ 5$$

then tell it what to do with them (+). So in Reverse-Polish we would have:

$$1 \ 5 \ +$$

RA is Reverse-Polish (and stack oriented). So remember, you have to tell RA what numbers to use, and then what to do with them.

$$6 \ 3 \ -$$

would be the same as $6 - 3$. You'll soon get the hang of it.

Currently, RA only supports integer math. That is, RA does not understand:

$$3.2 \ 1.9 \ +$$

Actually it does but not in the way you would think. Future releases of RA will deal with real math. The best you could do now is:

$$32 \ 19 \ + \ 10 \ /$$

be equal to 5.

RA™ strings are maintained on a stack as well. Literal strings however are not.

NOTE: RA™ numbers are stored as LONGINTs and Strings are stored as Str255s.

RA™ source files can be created in the 'Input' window, or by a text editor. RA™ will let you run either type of file. The file consists of text strings which represent code to be executed. The format of the code is straightforward. Spaces and cr/lfs are not critical. A source program must be terminated by a '\$' character. Any subroutines that are used in the program must be defined after the first '\$' character and before a final closing '\$'.

That is:

```
<program>
$
<subroutines>
$
```

Given this background we can now deal with the actual commands found in RA:

MATH OPERATORS --

+	Addition (not unary positive)
-	Subtraction (not unary negative)
*	Multiplication
/	Integer Division
\	Modulo

Examples:

5 7 +	12
3 4 5 + +	12
3 4 + 5 +	12
3 +	<error -- not enough numbers>
5 7 -	-2
3 1 -	2
9 1 3 - -	11
0 7 -	-7 -- this is the only way to get a negative number
7 -	<error -- not enough numbers>
1 0 *	0
3 2 * 1 *	6
3 4 3 **	36
6 2 /	3
12 2 /	6
12 15 /	0
7 3 /	2
7 0 /	<error -- division by zero>
6 3 \	0
5 3 \	2
5 0 \	<error -- modulo division >
4 5 + 2 - 3 2 /	1

BOOLEAN (LOGICAL) OPERATORS --

<	less than
>	greater than
=	equal
¥	AND
ß	NOT
Δ	OR

Examples:

1 3 <	-1 (true)
4 3 <	0 (false)
3 4 + 2 >	-1 (true -- 7 > 2)
1 3 =	0
1 1 =	-1
1 3 = 0 =	-1 (true-- 1 not equal to 3)
4 6 < 1 >	0 (false -- 1 not > 1)
0 1 - 0 1 - ¥	-1
0 1 - 0 ¥	0
0 1 - ß	0
0 ß	-1
0 1 - 0 Δ	-1
0 1 - 0 1 - Δ	-1
0 0 Δ	0
1 3 = ß	-1 (true-- 1 not equal to 3)

CONDITIONALS --

A RA conditional is denoted by the form:

<code> <test> [<code to execute if true>]

[] delimits the the code which is executed if the top number on the stack is true (-1).

Example:

0 1 - ["True"] will print True.

0 ["False"] will not print False.

4 3 > ["4 is greater 3"]

 will print 4 is greater 3.

4 3 > ["4>3" 3 4 > ["3>4"]]

 will print 4>3.

1 3 < 5 0 > ¥ ["1<3 and 5>0"]

 will print 1<3 and 5>0.

LOOPS --

() Delimits the code to loop.
~ Forces the loop to exit if the number on the top of the stack is false (0).

Examples:

("HI") An endless loop which will continuously print HI.

(1 3 < ~) An endless loop. The ~ tests the top number on the stack. In this case it would be -1 -- since 1 is not > 3.

(3 1 < ~) Would exit the loop.

(3 1 < ~ "3 is less than 1")
Would exit the loop without printing 3 is less than 1.

(3 1 > ["3>1" 1 3 > ~])
Would print 3>1 and then exit.

100 A: (A. 1 - A: A. ~)
Would loop 99 times. And exit when top of stack is = 0.

VARIABLES:

RA allows you to store values in variables. RA recognizes 26 variables named 'A' thru 'Z'. This may seem to be restrictive, but RA allows for the use of pointers, indirection, etc.

WARNING: The use of pointers and handles may destroy data if care is not exercised. Be sure you know what you are doing and understand pointers, handles and how to use them on the Mac. Always use a backup copy of your entire disk (not just the program you are working on.)

NOTE: All RA™ memory addresses and variables are relative to the location of Ra™ variable 'A'. That is variable 'B' is in RA™ memory location 1, 'C' in 2, etc.

All RA variables are global if they occur within the main body of the program. Any variables used within a subroutine (see same), are local to the subroutine and will not conflict with the same variable used outside the subroutine.

A thru Z	A variable.
:	Store the 2nd number on the stack in the location specified by the top number on the stack.
.	Retrieve the value from the memory location specified by the top number on the stack
..	Retrieve from pointer
...	Retrieve from handle.

Examples:

1 A:	Store 1 in variable named 'A'.
A.	Retrieve value in variable named 'A'.
3 A 2 + :	Store 3 in variable 'A' offset by 2 (equivalent to variable 'C'.)
C.	Retrieve value in C (3 if above statement executed first).
2000 A: 3 2000:	
A..	Retrieve the number 3 from location 2000. Variable A is a pointer to location 2000.

Subroutines:

RA provides a subroutine/subprocedure. Subroutines are useful for coding routines which can be used over and over. Up to 26 Subroutines can be defined in a given program. A subroutine consists of two parts the actual subroutine definition and the subroutine invocation of use. NOTE: Any variable used within a subroutine is local to the subroutine and does not conflict with the same variable name used outside the subroutine.

subroutine definitions must occur at the end of the main program and following the first terminator character.

The syntax for a subroutine Definition is as follows:

```
&<subroutine name> <code> `
```

where <subroutine name> is 'A' thru 'Z'. Note subroutine names are separate from variable names. That is, you can have a variable named 'C' and a subroutine named 'C' in the same program without conflict.

where <code> is any RA statement or another subroutine invocation. That is, a subroutine definition may invoke a subroutine in turn. NOTE: RA supports recursive subroutine calls. <code> may also be a formal subroutine parameter.

A RA formal subroutine parameter consists of:
<number>%. <number> specifies which actual parameter in the invocation will be used in place of the formal parameter.

The syntax of a RA subroutine Invokation is as follows:

```
#<subroutine name>,<actual parameters>;
```

where <subroutine name> is 'A' thru 'Z' and specifies which subroutine definition will be used.

<actual parameters> are separated by commas (,) and may include any RA statements (including other subroutine invocations).

Examples:

#A,"hello"; would print 'hello', given
&A 1% ` as the definition.

#B,10,2,36,4;
&B 1% 3% + ! ` would print 46.

#C,1,"HI ",5;
&C 2% 1% 3% + ! ; would print 'HI 6'.

Example of Global/Local variables

3 A: #A,100; A. ! will print 3 from global variable
'A'.

&A 1% A: ` stores 100 in the local variable 'A' of
subroutine A.

STRINGS:

RA™ allows the user to handle strings as well as numbers. All RA™ strings are of type Str255, unless they are literals (see I/O).

In addition to numeric variables, RA™ allows you to use string variables. As in numeric variables RA™ only supports 26 string variables, 'A' thru 'Z'. NOTE: There is no conflict between numeric variables, string variables and subroutine names using the same name.

†<string>†	defines a string which is pushed onto the string stack.
††	prints the top string off of the string stack.
†∂	stores the top element of the string stack on the address located as the top element of the number stack.
∂	pushes the string found at the address located as the top element of the number stack onto the string stack.

Examples:

†test†	pushes the string "test" onto the string stack.
††	would print "test" in the output window, assuming the previous push.
†test† A †∂	saves the string "test" in the string variable 'A'.
A∂	would push the contents of string variable 'A' onto the string stack.
††	Would print "test" given the above sequence.

†test† A∂	Saves "test" in string variable 'A'.
1 A:	Saves 1 in numeric variable 'A'.
#A;	Invokes subroutine 'A'.
&A "hi"~	Would print "hi".
A. !	Would print 1.
A∂ ††	Would print "test".

I/O

RA™ supports several forms of I/O, both string and numeric.

!	Prints the top element on the numeric stack.
!§	Prints the top element on the numeric stack as an ASCII character.
"!"	Forces a cr/lf
?	Queries the keyboard for a numeric entry. Input terminates on the first non-numeric character, or a cr.
?§	Queries the keyboard for a single ASCII character.
"<string>"	Prints the imbedded string.
††	Prints the top element of the string stack.
Ω	Positions the cursor to the location located as the top 2 numbers on the numeric stack. (top element = y, next element = x)
@	Clear output window

Examples:

3 !	Prints 3
7 !§	Prints a <bel> (ctrl-g)
"!"	Prints a cr/lf
"hello"	Prints "hello".
†hello† ††	Prints "hello" off of the string stack.
10 20 Ω "hi"	Prints "hi" at x,y = 10,20.
@	Clear window

PROGRAMATIC CONTROL OF OUTPUT WINDOW:

Even though RA™ does not allow you to manipulate the output window from the application, it does let you do so programmatically. The following commands let you change the output window from your programs:

@	Clear output window
↵	Move window to x,y located as top two elements on the numeric stack.
∫	Resizes the output window to h,v size located as top two elements on the numeric stack.
√	Either hides or shows the output window depending upon the top number on the stack. If -1 (TRUE) then the window is shown if 0 (FALSE) the window is hidden.
≈	Initializes the output window to its default size/location.

Examples:

@	Clears the window
30 30 ↵	Moves window to x,y = 30,30
20 100 ∫	Resizes window to 20h 100v
0 1 - √	Hides the output window
0 √	Shows the window
≈	Reinitializes the window.

MACINTOSH TOOLBOX CALLS:

RA™ allows you to use the Macintosh toolbox. RA™'s toolbox interface is somewhat simpler to use than Pascal's, C's or most other languages since RA™ worries about making certain that the data types of the toolbox call are correct. That is, you as the user need not worry about INTEGERS, LONGINTs, VARs, PTRs, HANDLEs, etc. RA™ will also make sure that the correct number of parameters are passed. One current drawback of the RA™ interface is that it does not recognize symbolic TrapNames, but only deals with the actual trap number (Axxx.)

The syntax for invoking a toolbox call is:

<params> <trap address> ¶

Any function result will be returned on either the numeric or string stack depending upon the actual trap invoked.

To use the toolbox interface, load all of the parameters on the proper stack(s) load the trap address on the numeric stack and then invoke it with a ¶.

NOTE: Any trap which requires a routine selector must have the routine selector pushed onto the stack before the trap address.

WARNING:

The Macintosh Toolbox may destroy the data on your disk if you do not know what you are doing. You should be familiar with the contents of Inside Macintosh or Macintosh Revealed. Always use a backup of your entire disk.

NOTE: Not all traps have been implemented in this release. More will be available with the next one. This version of RA™ (0.1) only implements the following traps:

<i>FreeMem</i>	<i>A01C</i>
<i>DisposPtr</i>	<i>A01F</i>
<i>SetPtrSize</i>	<i>A020</i>
<i>GetPtrSize</i>	<i>A021</i>

<i>DisposHandle</i>	<i>A023</i>
<i>SetHandleSize</i>	<i>A024</i>
<i>GetHandleSize</i>	<i>A025</i>
<i>ReallocHandle</i>	<i>A027</i>
<i>HLock</i>	<i>A029</i>
<i>HUnlock</i>	<i>A02A</i>
<i>EmptyHandle</i>	<i>A02B</i>
<i>SetApplLimit</i>	<i>A02D</i>
<i>BlockMove</i>	<i>A02E</i>
<i>SetDateTime</i>	<i>A03A</i>
<i>EqualString</i>	<i>A03C</i>
<i>ReservMem</i>	<i>A040</i>
<i>HPurge</i>	<i>A049</i>
<i>HNoPurge</i>	<i>A04A</i>
<i>CompactMem</i>	<i>A04C</i>
<i>PurgeMem</i>	<i>A04D</i>
<i>MaxMem</i>	<i>A11D</i>
<i>NewHandle</i>	<i>A122</i>
<i>RecoverHandle</i>	<i>A128</i>
<i>BitAnd</i>	<i>A858</i>
<i>BitXOr</i>	<i>A859</i>
<i>BitNot</i>	<i>A85A</i>
<i>BitOr</i>	<i>A85B</i>
<i>BitShift</i>	<i>A85C</i>
<i>BitTst</i>	<i>A85D</i>
<i>BitSet</i>	<i>A85E</i>
<i>BitClr</i>	<i>A85F</i>
<i>UprString</i>	<i>A854</i>
<i>Random</i>	<i>A861</i>
<i>StuffHex</i>	<i>A866</i>
<i>LongMul</i>	<i>A867</i>
<i>FixMul</i>	<i>A868</i>
<i>FixRatio</i>	<i>A869</i>
<i>HiWord</i>	<i>A86A</i>
<i>LoWord</i>	<i>A86B</i>
<i>FixRound</i>	<i>A86C</i>
<i>SetPt</i>	<i>A880</i>
<i>Secs2Date</i>	<i>A9C6</i>
<i>Date2Secs</i>	<i>A9C7</i>
<i>IUDateString</i>	<i>A9ED</i>
<i>IUDateString</i>	<i>A9ED</i>

NumToString
StringToNum

A9EE
A9EE

MISCELLANEOUS COMMANDS

The following programatic commands are also available from within the RA™ language.

- Interpret all input as hex. Output remains decimal. Any input hex value must be preceeded by a 0.
- ° Interpret all input as decimal.
- | comment to next | or cr/lf.
} Toggle 'Instruction Select' between the braces.
- μ Pop top item off numeric stack.

THINGS TO COME

The upcoming versions of RA™ should have more traps available, the interpreter should be a bit faster, and if we can come up with a clean way of doing it, a record structure. Of course, some of the bugs should disappear also.

Again, if you would like to see something in RA™ or, have any problems or suggestions, let me know.

Ultimately, full access to the toolbox will be available and if we get enough interest a compiler to go along with the interpreter. An object-oriented RA™? Who knows. The more we hear from you, the more we'll do.

We realize RA™ has its shortcomings. RA™ however, is in its infancy and will grow.

THANKS
TO

LOFTUS BECKER -- for suggesting a way of dealing with the traps. Although his method is not currently being used, we are giving it further thought. (Ver.0.1)