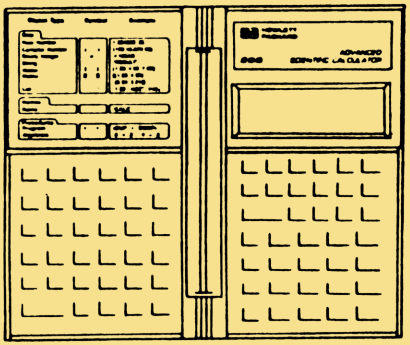
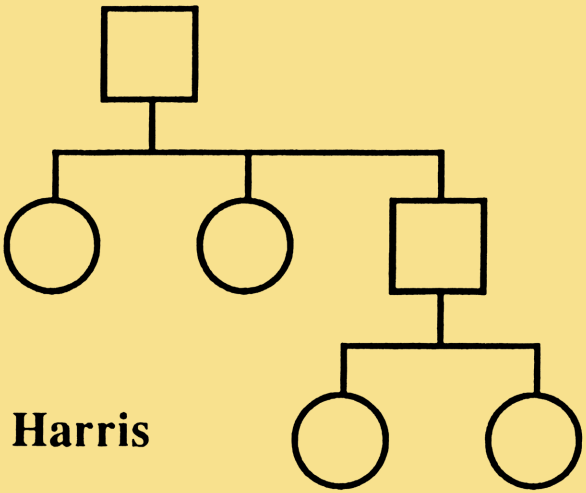
F O X

P R O J E C T



Memory Management Solutions for the HP28S

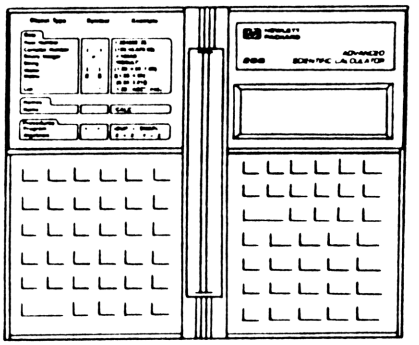


By
Martin D. Harris

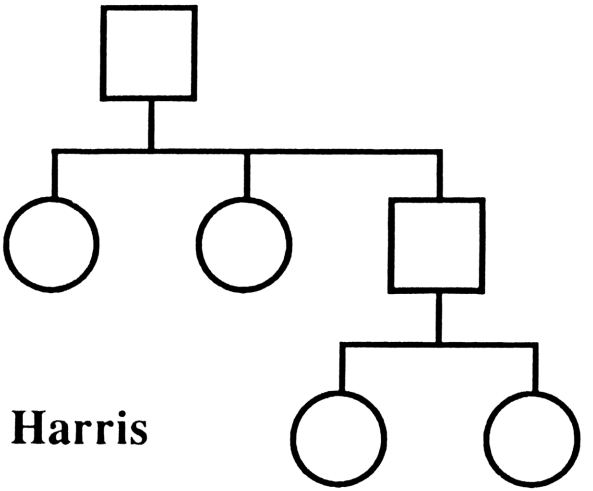
SECOND EDITION

F O X

P R O J E C T



Memory Management Solutions for the HP28S



By
Martin D. Harris

SECOND EDITION

FOX Project:
Memory Management Solutions for the HP28S
Second Edition

About the Author. Martin D. Harris is an undergraduate Electronic Engineering student at California Polytechnic State University, San Luis Obispo. His calculator programming experience includes the HP15C, HP28C, HP28S, and HP42S. He is an active member of the National Society of Black Engineers.

About the Publication. This work was published and distrubuted by the author. For a list of stores that sell FOX Project write to Martin D. Harris, 778 Boysen Ave #21, San Luis Obispo, CA 93405; or call 805-544-3602 or 213-295-2230.

AUTHOR'S NOTE

FOX Project, in its entirety, is a complete solution to memory management on the HP28S. At first glance, it might seem large and cumbersome. After all, it does take up about two kilobytes of memory and includes 16 programs. However, you don't have to use all of the programs. Detailed descriptions help you decide which programs you need. And there are practical exercises demonstrating how to use the programs to manage your memory. The important thing to keep in mind is that FOX was designed to save you time by providing efficient, clever programs to accomplish those tasks that would otherwise require several keystrokes and careful planning. The time you spend entering the programs and learning how to use them may seem like a lot at first but in the long run FOX pays for itself.

This material is based upon my three years' experience with the HP28S. I embarked upon this project with the intention of solving the difficulties of managing memory on the HP28S -- moving several variables at a time from one directory to another, renaming directories, locating variables, and purging non-empty directories -- using programs and without using SYSEVAL. The book is partitioned into five sections numbered zero through four. Section 0 reviews the fundamentals of user memory (familiarity with creating variables, creating directories, and entering programs is prerequisite to Section 0). Section 1 introduces and defines FOX Project. Section 2 describes the programs. Section 3 presents step-by-step exercises and some tips on how to enter the programs. Section 4 contains program listings. The examples in Section 2 and interweaved in the text of other sections are primarily for thinking through rather than working through. The idea is that refraining from punching buttons gives you a chance to focus on the concept first, before considering the mechanics of applying it. Section 3, however, has step-by-step exercises to give you practice using FOX to solve memory management problems. So the examples are useful for gaining a conceptual understanding of FOX and the exercises are useful for gaining a working knowledge of FOX. After reading Section 2 and after completing Section 3, feel free to return to Section 2 and work the examples to get even better acquainted with FOX.

Were there any changes to the first edition? Most definitely! Users of the first edition will find in this second edition some significant improvements: 1) the programs are easier to enter because the "bat" (character code 134) is not used; 2) in response to feedback from the field, I have devoted an entire section to step-by-step exercises; 3) the program set has been simplified by the elimination of four programs (BAT, D→S, D→M, and D→X) and at the same time the program set has been enhanced by the addition of four powerful programs (HUNT, RENM, PTRE, and ZAP). To perform D→S, D→M, or D→X use the VARS command followed by V→S, V→M, or V→X, respectively. All of the programs have been streamlined. Overall, FOX is now more efficient and more useful.

I would like to thank: My mother Elnora Crowder for her wisdom; Maya Gitelson and Percival Parks for their encouraging words; Raphael Hernadez for his time; Debbie Gibson and Ken Johnson for the cover design; and the staff of Chicago Public Schools, Mid-City Gifted Program Office, for their support. I am very grateful to these people and the many others who motivated, advised, and assisted me in this endeavor. Special thanks to illustrator Ray Tong and proof readers Azizi Jones and Bruce Newman, without whom the second edition would not have been possible.

This book is dedicated to Ms. Mary Owens, an outstanding scholar and inspirational teacher who coached me in writing.

All proceeds go towards printing costs and towards encouraging me to write more. For comments and correspondence in general, please write 778 Boysen Ave #21, San Luis Obispo, CA 93405, or call 805-544-3602 or 213-295-2230. Feedback is very much welcomed.

FOX Project: Quick, Smart, Clever. Enjoy!

Martin D. Harris
September 1991

CONTENTS

USER MEMORY BASICS

This section briefly explains menus, variables, and directories. If, after reading this section, you feel comfortable with the concepts presented, then you may proceed confidently onto the next section. But if not, then review the Owner's Manual and become more familiar with the HP28S before going on. The following topics will be covered:

- Menu Keys
- USER Menu
- Variables as User-Defined Commands
- Directories as Sets of Variables
- Current Path
- Ghost Names

What are Menu Keys?

Unlike many calculators, the HP28S has more commands than it has keys. It has over 200 built-in commands but it has only 72 keys, half of which are for letters and numbers. That leaves 36 keys for commands. This apparent shortage of keys is taken care of by the menu keys which are the first row of keys immediately below the display.

Menu keys are keys that can have several meanings. They are not limited to representing one or two commands. The division key represents only two commands: "divide" and, when used with the shift key, "invert." Just as painted-on symbols (÷ and 1/X) depict the meaning of the division key, menu labels depict the meaning of the menu keys.

Menu labels appear at the bottom of the calculator's display when activated by a menu selection key. The keys MODE, TRIG, LOGS, ARRAY, BINARY, USER, and CUSTOM are all examples of menu selection keys. Selecting different menus gives the menu keys different meanings. The menu

labels show what the menu keys mean for the currently selected menu. Consider the LOGS menu. It contains commands useful for calculating logarithms. When the LOGS menu is activated, six menu labels appear at the bottom of the calculator's display. Each label has the name of a logarithm-related command: LOG, ALOG, LN, EXP, LNP1, and EXPM appear on the last line of the display. So pressing the menu key (immediately below the display) corresponding to ALOG, for example, commands the calculator to take the antilogarithm of a number. Only six menu labels can be displayed at a time. For menus having more than six commands, the NEXT key causes the next six commands to be displayed. Each set of six menu labels is called a row. So when the LOGS menu is selected, pressing NEXT causes the next row of logarithm commands to be displayed, namely SINH, ASINH, COSH, ACOSH, TANH, and ATANH.

What is the USER Menu?

The USER menu is distinguished from the other menus by the fact that it contains commands that are defined by you, the user. These user-defined commands are generally called variables. Variables allow you to taylor your calculator to suit your specific needs.

Here is an example. The cotangent function is not a built-in function of the HP28S. But you can create a program to compute the cotangent: « TAN INV » [ENTER] 'COT [STO]. The STO command in this example creates a variable named 'COT'. When the USER menu is activated, pressing the menu key that corresponds to menu label 'COT' executes the program 'COT'. Furthermore, you can use the CUSTOM menu to display variables (user-defined commands) and built-in commands together, in the same menu. The following does just that: { COT SIN COS ATAN } [ENTER] [MENU]. The MENU command (in the MEMORY menu) makes both built-in commands and user-defined commands easily accessible through the CUSTOM menu.

Here's how variables work in general. Typing the following sequence defines a variable called 'G': 32.2 'G [STO]. When the USER menu is activated, 'G' appears as a menu label. Pressing the menu key directly below that label can be thought of as "commanding" the calculator to put the contents of 'G' onto the stack. If 'G' were a program, pressing the menu key that corresponds to label 'G' would in effect "command" the calculator to execute the program. In either case, the variable is used to command the calculator to put something onto the

stack or to execute a program. The contents of the variable specifies either what to put onto the stack or what set of instructions to perform.

What are Directories?

A directory is a set of variables. Directories allow you to organize your variables into sets (groups) by course, chapter, type, and so on. You can define directories within directories to make a hierarchy of variables. Such a hierarchy is called a memory tree or simply a tree. FIG 0.1 shows an example of a tree. The boxed names represent directories and the circled names represent variables.

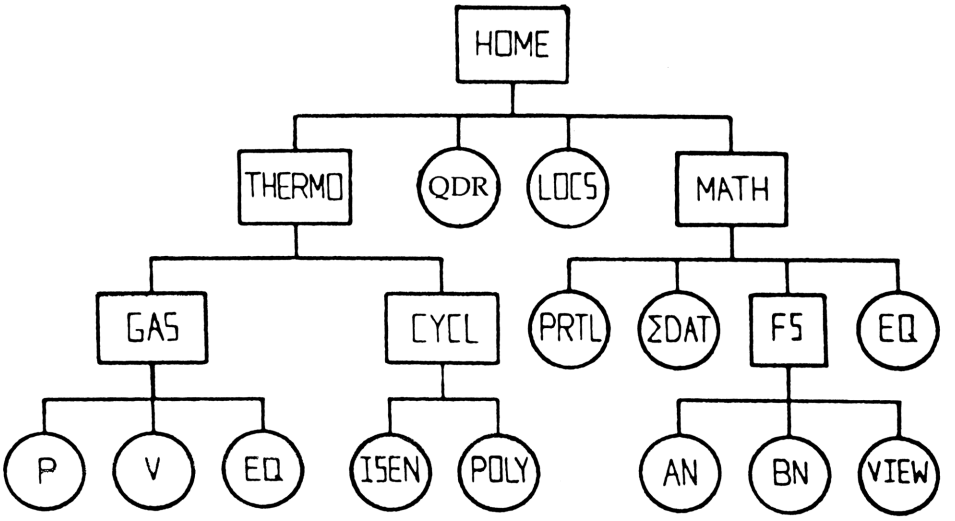


FIG 0.1: Example Memory Tree.

HOME can be thought of as the "root and trunk" of the memory tree. Directories can be thought of as a "branches," subdirectories as "limbs," and

variables as "leaves" of the memory tree. HOME is permanent and from it stem all of the variables and directories of user memory.

The following sequence creates a directory called 'PHYS' for Physics in the HOME directory: [HOME] 'PHYS [CRDIR]. The HOME and CRDIR commands are in the MEMORY menu. When the USER menu is activated, 'PHYS' appears as a menu label. Pressing the menu key corresponding to 'PHYS' changes the current directory from HOME to 'PHYS', moving you from the directory whose path is { HOME } to the directory whose path is { HOME PHYS }. This is similar to moving from one command menu to another. But there is one big difference: the directory 'PHYS' is contained within HOME. Put another way, 'PHYS' is a subdirectory of HOME. Put yet another way, HOME is a set of variables of which 'PHYS' is a subset. Additionally, HOME is said to be the parent directory of PHYS.

Only one directory at a time may be viewed in the USER menu. This allows you to focus on one set of variables at a time. Whenever a directory has its contents visible in the USER menu, that directory is the current directory. So when in FIG 0.1 'MATH' is the current directory 'PRTL', 'ΣDAT', 'FS', and 'EQ' appear on menu labels in the USER menu. Other names such as 'QDR', 'GAS', and 'VIEW' (to name just a few) do not appear because 'MATH' is not their parent directory.

While only one directory at a time may be viewed in the USER menu, all variables and directories on the current path are accessible. (The current path is simply the path of the current directory). Suppose again that the USER menu looks like FIG 0.1 and 'MATH' is the current directory. Certainly you can recall variables 'PRTL', 'ΣDAT', and 'EQ' and evaluate directory 'FS'. But less obvious is the fact that you can also recall variables 'QDR' and 'LOCS' and evaluate directories 'THERMO' and 'MATH'. The commands RCL (recall) and EVAL (evaluate) search the entire current path for a name. First they check the current directory, then the parent directory, and then the parent's parent directory and so on.

EVAL is applicable to both variables and directories. RCL is applicable to variables only. Directories cannot be recalled with RCL, hence the need for FOX Project. (The PRVAR, like RCL, searches the entire current path and is applicable to variables only.)

The variables and directories accessible from the current directory are those that are defined in each directory of the current path. The PATH command puts onto the stack the path of the current directory. This path is a list of directories that lead up to the current directory. Does this sound confusing? Well, it is difficult to understand. But hopefully these examples will help. The path of 'MATH' is { HOME MATH }, so all the variables of 'MATH' ('PRTL', 'ΣDAT', 'FS', and 'EQ') and HOME ('THERMO', 'QUAD', 'LOCS', and 'MATH') are accessible. The path of 'GAS' is { HOME THERMO GAS }, so all the variables of 'GAS' ('P', 'V', and 'EQ'), 'THERMO' ('GAS' and 'CYCL'), and 'HOME' ('THERMO', 'QUAD', 'LOCS', and 'MATH') are accessible. Again, the term "accessible" in this context means subject to use with RCL, PRVAR, or EVAL.

What Does All of this Mean?

User memory consists of the variables and directories of the USER menu. Use the USER menu to create your own commands. Use directories to arrange your commands into sets and sets within sets. The most challenging aspects of managing memory on the HP28S are 1) understanding directories, 2) choosing an intelligent arrangement of directories, 3) choosing names for variables and directories, and 4) rearranging and renaming variables and directories to suit your changing needs. This section helps with the first item. The FOX Project programs help with the last item.

Where Do I Go From Here?

Admittedly, this has been a very brief review. If you understand how to use menus, variables, and directories and how to enter programs, then you are ready to read the program descriptions, enter the FOX programs into your HP28S, and use FOX Project to manage your user memory. Otherwise, it would be best to review chapters 3, 4, and 20 ("Using Variables," "Repeating Calculations," and "Memory") of the HP28S Owner's Manual.

SECTION 1

INTRODUCTION

1.1 General Description

FOX Project is a set of programs that automate memory management on the Hewlett-Packard HP28S Scientific Calculator. Basically, it facilitates the locating, recalling, storing, and purging of variables and especially that of non-empty subdirectories. The programs were designed with three primary objectives: 1) to prevent the accidental erasure of variables and directories; 2) to minimize memory requirements; and 3) to maximize execution speed. FOX Project gives HP28S users safe, efficient tools for use individually or as parts of customized programs.

FOX Project is most recommended for those familiar with creating directories, entering programs, and evaluating programs. Those who have experienced the frustration of moving a subdirectory (and all of its contents) from one parent directory to another parent directory are more than ready to dive right into this project. And those who have hesitated to experiment with alternate ways of setting up their directories because of the difficulties of renaming, erasing, and replicating directories, will find this project of great value.

1.2 Motivation

Memory size and memory organization set the HP28S apart from the vast majority of hand-held calculators. Having 32 kilobytes of user memory makes external memory devices pretty much unnecessary, though such devices would be convenient for sharing programs. The HP28S does not support any peripheral input devices (such as the card reader available for the HP41 or the ROM card available for the HP48SX). This has two implications: 1) the keyboard is the only way to input programs, and 2) user memory must be carefully conserved.

Memory organization on the HP28S is similar to MS-DOS file organization on personal computers. The HP28S has variables (analogous to files), directories, and subdirectories. The entire user memory is arranged like a tree with variables (leaves) stored (grafted) in subdirectories (limbs), which are themselves stored (grafted) in directories (branches). And similar to MS-DOS, the built-in functions for managing memory operate only on variables and empty subdirectories. These built-in functions -- STO (store), RCL (recall), PURGE, and CLUSR (clear user memory) -- will result in an error if given the name of a non-empty subdirectory as the argument for STO, RCL, and PURGE, or if a non-empty subdirectory exists in the current directory when trying to use CLUSR. If one wishes to 'purge' an entire subdirectory or to 'recall' the contents of a subdirectory onto the stack, one would have to carefully maneuver in and out of subdirectories and repetitively execute the built-in memory management functions. FOX Project tremendously automates such operations.

1.3 Terms and Symbols

Brackets "[" and "]" will be used to emphasize the typing of a menu key or any key in general on the HP28S keyboard (e.g. [ENTER] , [STO] , [PATH] , and [VARS]).

The arrow is an important HP28S symbol and is used frequently in FOX programs as a character in names and as a command. An arrow "→" will be used in this text to depict the arrow key (SHIFT-U) on the HP28S keyboard (e.g. V→S and V→M).

A "name" is a variable or directory that is defined in user memory (and therefore visible on a menu label when the USER menu is active). In a different sense, a "name" is a sequence of characters enclosed in single quotes.

Variables and directories that are on the current path but not in the current directory are called "ghost" names. Refer to FIG 0.1. When 'CYCL' is the current directory, 'GAS', 'CYCL', 'THERMO', 'QDR', 'LOCS', and 'MATH' are ghosts. Although they do not appear in the 'CYCL' directory with 'ISEN' and 'POLY', they are still accessible via RCL, PRVAR, or EVAL.

The terms "program," "variable," "directory," "subdirectory," "current directory," "current path," and "parent directory" will be used according to their definitions given in the glossary of the HP28S Reference Manual. It would

be worthwhile to review those terms now. The terms "directory" and "subdirectory" are synonyms and will be used interchangeably.

The term "name" is the most significant. Almost everywhere it appears in this text, "name" can be replaced by the words "variable or subdirectory." Additionally, "names" (plural) can be replaced by the words "variables and subdirectories." So "locate a name" means "locate a variable or subdirectory" and "all names are purged" means "all variables and subdirectories are purged." The Only exception is when "name" refers to a sequence of characters in single quotes (e.g. 'A', 'VALU1').

1.4 Special Features

Preserving Order. Names recalled from one directory and subsequently stored into another directory appear on the menu labels in their original order.

Preventing Overwriting. The storing operation ends with an error message if an attempt is made to store a set of names into a directory containing variables or subdirectories that match those names. The user may then rename or purge the existing variable or subdirectory defined in the current directory and resume with the operation. The renaming operation ends if the old name does not exist or if the new name matches a name that already exists in the current directory. These are just two examples of how FOX guards against careless mistakes.

Handling Ghost Names. A name that is not defined in the current directory may be accessed via RCL, PRVAR, or EVAL if the name exists on the current path. FOX can help locate and purge ghost names. This is useful because ghost names can foil plotting and solving operations.

These are just some of the features that make FOX Project a robust solution to managing memory on the HP28S. The FOX Project programs work together and, upon encountering non-ideal conditions, safely end without corrupting user memory or the stack. Enjoy!

PROGRAM DESCRIPTIONS

This section contains diagrams and text that describe the FOX Project programs. TABLE 2.1 shows a complete listing of the program names and operations. In the examples that follow, the notation {. . . P } means that the directory 'P' can be anywhere. For example, the actual path might be { HOME P } or { HOME TEST P }.

TABLE 2.1: FOX Project Program Description and Listing References.

#	NAME	OPERATION	DESCR. page	LISTING page
1	MM	Interface	2-2	4-2
2	MSET	Interface List	2-2	4-2
3	ALRM	Audio Prompt	2-3	4-2
4	PL	Locate Parent Directory	2-3	4-2
5	DL	Locate a Directory	2-3	4-2
6	P?	Identify Current Path	2-3	4-2
7	PTRE	Create 'P' Tree	2-3	4-3
8	V→S	Recall Names	2-4	4-3
9	V→M	Store Names	2-5	4-4
10	V→X	Purge Names	2-6	4-5
11	FIND	Locate a Name, Downwards	2-7	4-5
12	SEEK	Locate a Name, Downwards	2-8	4-5
13	HUNT	Locate a Variable, Upwards	2-8	4-6
14	RENM	Rename a Name	2-9	4-6
15	CLDIR	Clear Current Directory	2-10	4-7
16	ZAP	Purge a Name, Upwards	2-11	4-7

MM	Interface

• Uses the custom menu to display some FOX Project programs and some built-in commands; simply type 'MM' for "Memory Management" from any directory and the programs and built-in commands you need to effectively manage your memory are at your fingertips.

MSET	Interface List

• A list of programs and commands pertinent to memory management; the names listed in MSET appear in the custom menu as a result of using MM.

• May be modified to suit your own preferences; for example a) original set-up, b) suggestion #1, c) suggestion #2, and d) suggestion #3:

a)

P?	V→S	V→M	VARS	HUNT	PL
P?	RENM	V→X	FIND	DL	PL

b)

HOME	VARS	V→S	V→M	PL	P?
RENM	HUNT	FIND	V→X	PL	DL

c)

V→S	V→M	SEEK	V→X	PL	P?

d)

P?	V→S	V→M	RENM	HUNT	PL
P?	VARS	FIND	V→X	DL	PL

ALRM	Audio Prompt

- Beeps (if flag 51 is clear).

PL	Locate Parent Directory

- Moves up the current path one step; if the current directory is HOME then HOME remains the current directory.

DL	Locate a Directory

- Takes a directory path (list) from stack level one and evaluates it, moving from the current directory to the directory having the same path as the one given in stack level one.

P?	Identify Current Path

- Momentarily displays the path of the current directory (the current path), letting you know where you are in user memory.

PTRE	Create 'P' Tree

- Recreates the tree of FIG 2.1 in the HOME directory.

- Particularly useful for practicing FOX programs on the tree of FIG 2.1; after each experiment, execute PTRE to get 'P' tree back to normal for yet another experiment.

V→S	Recall Names

- Recalls from the current directory the contents of each name specified in a list in stack level one.

- If the list in stack level one contains names that are not defined in the current directory, then V→S will ignore them and recall only those names that are defined in the current directory .

- Returns to the stack a stack-coded directory; level one contains the number of successfully recalled names.

- Memory is not altered, only copied onto the stack.

EXAMPLE for V→S	before	after
TREE	FIG 2.1	same
PATH	{...PS}	same
STACK	{A B}	FIG 2.2

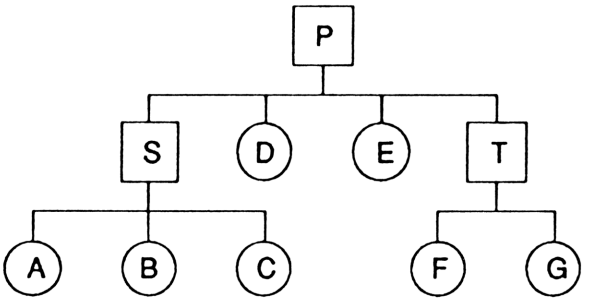


FIG 2.1: Initial Tree.

```
6: (2,1)
5: 'A'
4: (2,2)
3: 'B'
2: 'S'
1: 2
```

FIG 2.2: Stack-Coded Directory.

V→M	Store Names

- Stores the names coded on the stack into the current directory.

- If any of the names on the stack match those already present in the current directory, then the operation will stop and display an error message; you may then rename or purge the existing name and resume with the operation by re-running the program V→M.

- The stack must contain a stack-coded directory starting in level one (see Appendix); such is the output of V→S.

EXAMPLE for V→M	before	after
TREE	FIG 2.1	FIG 2.3
PATH	{...PT}	same
STACK	FIG 2.2	back to normal

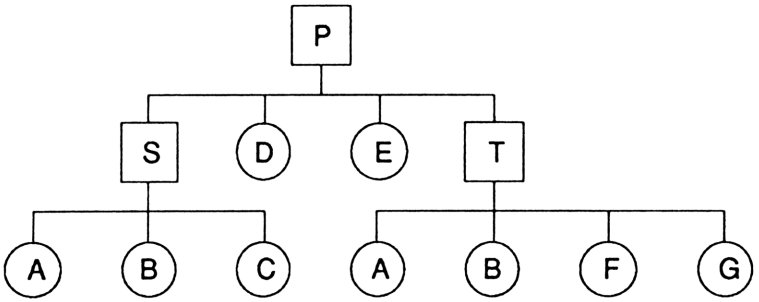


FIG 2.3: Result of Example for V→M.

V→X	Purge Names

• Purges each name specified in a list (in stack level one) from the current directory.

• To erase the entire HOME directory, and hence all of user memory, go to the HOME directory, select all of the names (e.g. by using the VARS command that is in the MEMORY menu), and then use V→X; this is the only way to erase all of user memory with FOX Project; using CLDIR from HOME will not erase anything (that's a safety feature).

EXAMPLE for V→X	before	after
TREE	FIG 2.1	FIG 2.4
PATH	{ . . . P }	same
STACK	{ S E }	back to normal

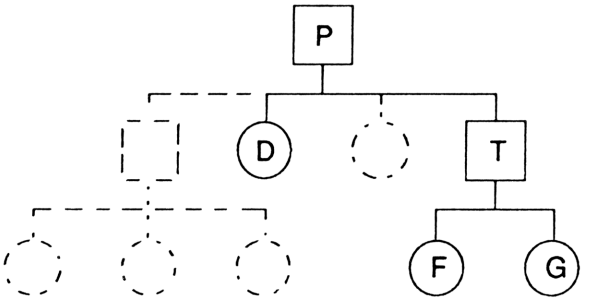
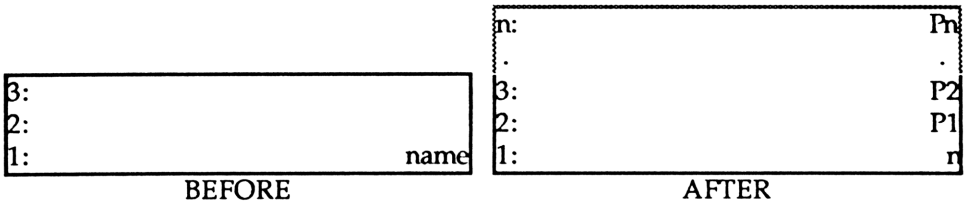


FIG 2.4: Result of Example for V→X.

FIND	Locate a Name, Downwards



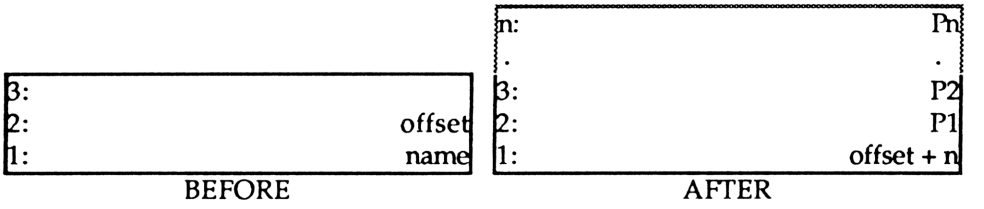
- Searches through the current directory and through all of its subdirectories for a name specified in stack level one.

- Returns to the stack the path of each directory that contains that name and returns to stack level one the number of occurrences of that name (the number of paths).

- Does not search above the current directory.

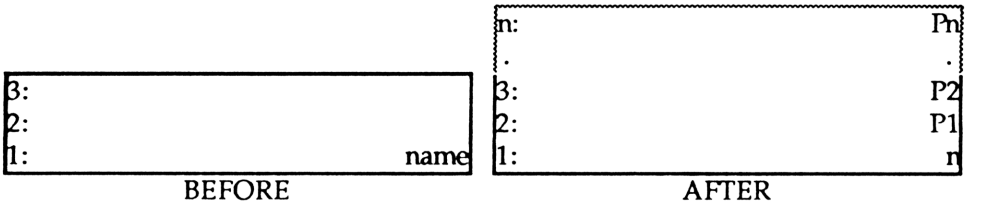
- To search through all of user memory, execute FIND from the HOME directory (this will report all locations of a name because HOME is the top-most directory in the hierarchy of directories).

SEEK	Locate a Name, Downwards



- A subprogram of FIND; does everything FIND does but requires that stack level two contains a number (use zero to initialize the counter); like using FIND, the name for which to search must be in stack level one.

HUNT	Locate a Name, Upwards



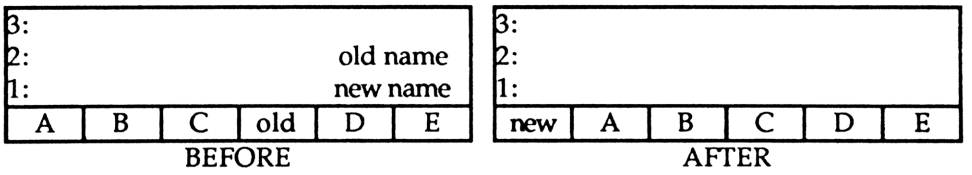
- Searches through each directory on the current path for a name specified in stack level one.

- Returns to the stack the path of each directory that contains that name and returns to stack level one the number of occurrences of that name (the number of paths).

- Does not search below the current directory (SEEK and FIND do search downwards).

- Particularly useful for reporting the location of ghost names.

RENM	Rename a Name



• Renames a name specified in stack level two with the new name specified in stack level one.

• Gives warning message "Not Ready" and ends without renaming if either the old name does not exist or the new name already exists in the current directory.

CLDIR	Clear Current Directory

- Purges all of the variables and directories in the current directory.

- Requires confirmation: press [Y] for "yes" after the "Delete?" message appears; pressing any key other than [Y] will cause the program to end without purging anything.

- Works like the CLUSR command but is more powerful because CLDIR purges the non-empty directories too!

- The directory's contents are purged but the directory itself remains defined in its parent.

- Works on any directory except HOME, thereby preventing the accidental erasure of the entire user memory in one shot; (read the description for V→X or simply do a System Reset if you want to clear the HOME directory.)

EXAMPLE for CLDIR	before	after
TREE	FIG 2.1	FIG 2.5
PATH	{...P}	same
STACK	anything	same

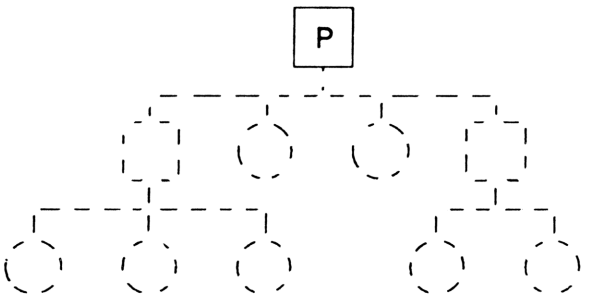


FIG 2.5: Result of Example for CLDIR.

ZAP	Purge a Variable, Upwards

- Purges a variable specified in stack level one from each directory on the current path.

- Particularly useful for erasing ghost variables which may foil plotting and solving operations.

- Will not erase directories; this safety feature mainly prevents the accidental erasure of the current directory (the directory from which ZAP is executed).

- Returns a number to stack level one; that number indicates the number of directories whose name matches the variable name that was given for "zapping"; (this happens because after it purges all occurrences of the variable, ZAP executes HUNT to report the locations of each directory whose name matches the name of the given variable); this way you know how many matching names remain on the current path and where they are; at that point, each matching name is a directory because all matching variables have been purged.

- Let's try that again; after purging variables, ZAP returns the number of remaining occurrences of the given name and the path of each occurrence; if the number is zero then that means that the name no longer exists anywhere on the current path; if the number is not zero then that means that the name given for "zapping" matches names of directories (on the current path); the number and locations of such directories shows up on the stack.

- See the examples on the next few pages.

EXAMPLE #1 for ZAP	before	after
TREE	FIG 2.1	FIG 2.6
PATH	{ . . . P S }	same
STACK	'E'	0

COMMENTS: If 'E - 5 / B' is stored in 'A', then evaluating (pressing EVAL twice after putting 'A' on the stack) will cause the calculator to plug in values for variables B and E. The calculator looks along the current path for these variables: looking first in the current directory, then in the parent directory, then in the parent's parent directory, and so on. When 'A' is evaluated while 'S' is the current directory, the calculator looks for 'B' and 'E' in 'S', finds 'B' but not 'E', then looks in 'P' for 'E' and finds 'E' there. So if 'B' is in 'S' as B=5 and 'E' is in 'P' as E=7, then evaluating 'A' yields A=(8-5/5)=7. This effect is great if 'E' is common and relevant to all equations and programs stored in directories 'S' and 'T' (as would be the case if 'E' were a physical constant for all of the Physics variables of directories 'S' and 'T' since 'E' is accessible from both of those directories). Otherwise it can be a problem. If you wish to have 'E' appear as a symbol when 'A' is evaluated, you must purge every occurrence of 'E' on the current path (which includes the current directory too). That is what ZAP does for you. Then evaluating 'A' (with 'A' ,'B', and 'E' as defined above) will return 'E - 1' instead of 7.

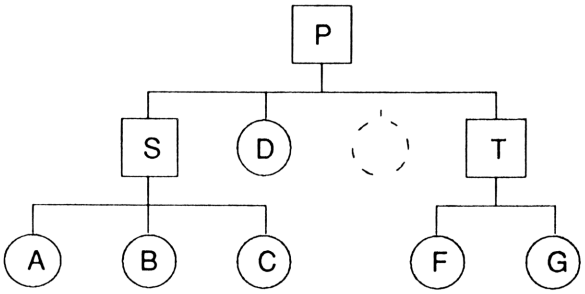


FIG 2.6: Result of Example #1 for ZAP.

EXAMPLE #2 for ZAP	before	after
TREE	FIG 2.1	same
PATH	{ . . . P S }	same
STACK	2:	2: { . . . P }
	1: 'T'	1: 1

COMMENTS: Suppose you have the equation 'T + 273' stored in 'C'. An attempt to evaluate 'C' results in a change of directories and possibly a corrupted stack or a "Too Few Arguments" error. This is because 'T' is on the current path and is a directory. Instead of purging 'T', ZAP reports that 'T' is the name of 1 directory on the current path and that 'T' is a subdirectory of directory 'P' (put another way, 'T' is located at { . . . P }). With this information you could go to directory 'P' (using DL) and either purge 'T' (using V→X) or rename 'T' (using RENM) to eliminate 'T' from the current path. That would free you to use 'T' in the equation. Or you could choose to leave 'T' alone and select a different variable name for use in the equation (e.g. 'TMP + 273').

EXAMPLE #3 for ZAP	before	after
TREE	FIG 2.1	same
PATH	{ . . . P S }	same
STACK	2:	2: { . . . }
	1: 'P'	1: 1

COMMENTS: Here an attempt was made to zap 'P'. This is not allowable since the current directory would be purged too. So instead ZAP just reports the location.

EXAMPLE #4 for ZAP	before	after
TREE	FIG 2.7	FIG 2.8
PATH	{ . . . G J R }	same
STACK	2:	2: { . . . G J }
	1: 'T'	1: 1

COMMENTS: A variable 'T' is purged from the current directory, a variable 'T' is purged from the parent's parent directory (two steps up the current path), and the location of directory 'T' is reported. 'T' at { . . . G W } is not included because it is not on the current path so it is not a ghost and hence is not accessible from directory 'G'.

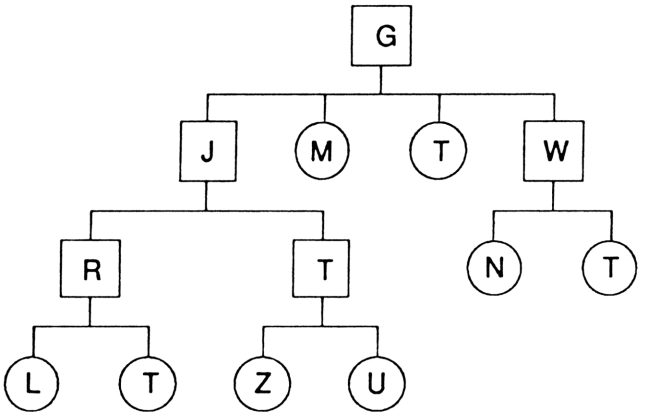


FIG 2.7: Before Example #4 for ZAP.

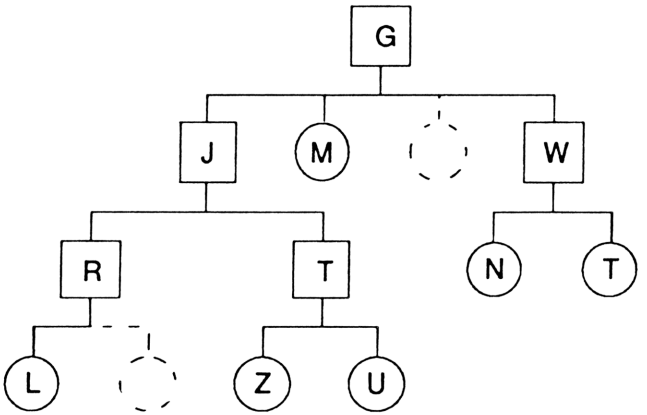


FIG 2.8: After Example #4 for ZAP.

EXERCISES

NOTE: It is recommended that you read through this section once briefly before attempting do the exercises.

3.1 Getting Started

The fastest way to get familiar with FOX is to practice using it on FIG 3.1 and FIG 2.1 as directed by the exercises of this section and by the examples of Section 2. The exercises of this section lead you to make changes to 'P' tree, the tree in FIG 3.1.

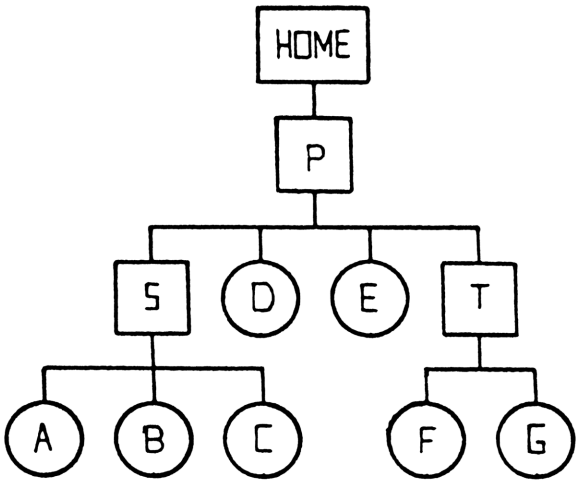


FIG 3.1: Initial Tree, 'P' Tree, that results from PTRE.

It would be nice to have a way to restore 'P' tree back to its original form for the purpose of recovering from a mistake, repeating an exercise, or just experimenting on your own. Well, program PTRE does just that. It allows you to experiment, mess-up, and then quickly get things back to normal. Use PTRE from any directory by typing PTRE and pressing ENTER and it will create 'P' tree in the HOME directory. (Before using PTRE, be sure no variable or directory is named 'P' in the HOME directory because PTRE will erase it.)

The programs V→S, V→M, and V→X are extensions of the built-in commands RCL (recall), STO (store), and PURGE, respectively. These built-in commands work only for variables and empty directories. The FOX programs go a few steps beyond these constraints, allowing greater flexibility in working with variables and directories (empty directories and non-empty directories alike).

For starters, whenever you want to use one of the programs, simply select the CUSTOM menu and press the menu key that corresponds to the name of the desired program. If the custom menu does not display names of FOX programs, then typing MM (followed by ENTER) will re-establish the custom menu to display them. Which program names appear in the custom menu is determined by MSET. The original definition of MSET will be used throughout the exercises of this section, but as you become familiar with FOX, feel free to modify MSET to contain whatever FOX program names and built-in commands (like HOME, VARS, and PATH) you use most often (see the description of MSET in Section 2 for some suggestions). FIG 3.2 shows what the custom menu looks like after using MM.

P?	V→S	V→M	VARS	HUNT	PL

a) CUSTOM (to see the first row)

P?	RENM	V→X	FIND	DL	PL

b) NEXT (to see the second row)

FIG 3.2: The CUSTOM menu after using MM.

The exercises use the following representations of keystrokes on the HP28S:

TEXT STYLE LEGEND FOR EXERCISES

Plain Text	=	type letters
Underlined Text	=	press key
Outlined Text	=	press menu key

3.2 Recalling Names

Typically, to recall a variable, one puts the variable's name in single quotes onto the stack and presses RCL from the keyboard. The content of the variable then appears in stack level one. That's fine, but an attempt to use RCL on a directory will result in a "Directory not Allowed" error.

Here's how V→S (Recall Names) solves this problem. V→S requires a list of names as the argument on the stack. The list can contain one name or several names. To recall a subdirectory, simply put its name in a list and use V→S as illustrated in Exercise #1.

EXERCISE #1

Step	Type	Result
1.	PTRE ENTER	Establishes FIG 3.1 in the HOME directory and makes HOME the current directory. Note: no quotes are used in typing PTRE.
2.	MM ENTER	Sets up the CUSTOM menu for Memory Management.
3.	ℙ ?	Momentarily displays the current path. The path should be { HOME } at this point.
4.	USER P ENTER	Makes 'P' the current directory. (Pressing the menu key labeled 'P' does the same thing and does not require ENTER.) The menu labels show names S, D, E, and T.

S	D	E	T		

USER menu after step 4.

5.	CUSTOM ℙ?	Momentarily displays the current path. The path should be { HOME P } at this point.
6.	USER { α 𝕊 G ENTER	Puts names 'S' and 'G' into a list on the stack. The Alpha Key (α) puts the cursor into Alpha Entry Mode, where pressing a menu key results in the writing of its label in the command line. ENTER moves the contents of the command line to the stack and adds the closing brace. The name 'S' may be either typed or selected from the USER menu. The name 'G', however, must be typed (from the left-hand keyboard) because it is not a part of the current directory (it does not appear on the USER menu labels). 'G' is included to illustrate how V→S ignores names that are not a part of the current directory. The stack contains the list { S G } and the menu labels show names S, D, E, and T.

1:					{S G}
S	D	E	T		

USER menu after step 6.

7.	CUSTOM 𝕍→𝕊	V→S takes the list { S G } and returns a stack-coded directory.

After step 7, a stack-coded directory appears on the stack. FIG Ex1.1 gives a graphical interpretation of the stack-coded directory. The triangle signifies that 'P' is a stack-coded directory rather than a regular directory. Note that the triangle 'P' of FIG Ex1.1 has one name branching from it whereas the square 'P' of FIG 3.1 has four names branching from it. User memory has not been altered as a result of using V→S. Only now the stack shows a portion of user memory.

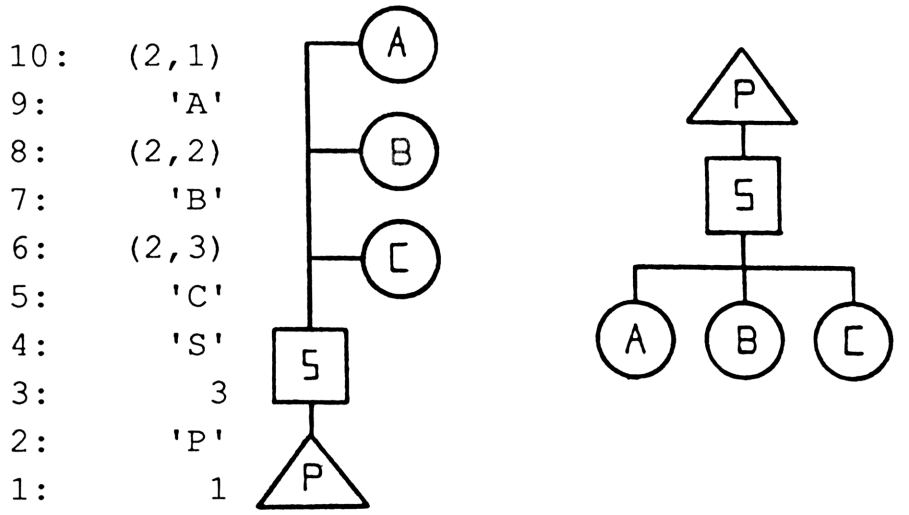


FIG Ex1.1: Graphical Interpretation of the stack-coded directory that results from Exercise #1.

The first item (stack level one) is the number of names that were actually recalled. The second item (stack level two) is the name of the directory from which V→S was executed. Together items one and two make up the header of the stack coded directory. The rest of the levels contain the recalled variables and directories.

V→S tells the calculator, "Recall these names from the current directory." In Exercise #1 the argument of V→S was the list { S G }. V→S attempted to recall both 'S' and 'G' but recognized that 'S' was a part of the current directory and that 'G' was not a part of the current directory. So, stack level one contains the number 1 instead of 2 even though two names were specified in the list. That is because only one of the names specified in the list actually exists in the current directory. V→S recalls only the names that are defined in the current directory, the directory from which V→S is executed. The header indicates how many names were recalled, hence the number of names that may be created with V→M, as we shall see.This ability to ignore foreign names is a safety feature, making V→S resilient to careless errors.

A better understanding of the stack-coded directory format will come with practice. It's okay to move on to the next exercise even if the idea of a stack-coded directory is not completely clear. It will become clearer as we forge ahead. For now, keep the results of Exercise #1 on the stack for use in the next exercise.

3.3 Storing Names

Typically, to store a variable, one puts the variable and a name onto the stack and presses STO. Consider how V→M works for a stack-coded directory. V→M creates the variables and directories of a stack-coded directory as illustrated in Exercise #2 and Exercise #3.

EXERCISE #2

NOTE: This exercise is to be performed immediately after Exercise #1 so that
TREE = FIG 3.1
PATH = { HOME P }
STACK = FIG Ex1.1

Step	Type	Result
1.	USER T	Goes to the 'T' directory.

F	G				

USER menu after step 1; path={ HOME P T }.

2.	CUSTOM V→M	Executes V→M from the CUSTOM menu.
3.	USER	Shows the USER menu. The menu labels show names S,F, and G.
4.	S	Goes to the 'S' Directory. The menu labels show names A, B, and C and the current path is { HOME P T S }

V→M created 'S' in 'T'. FIG Ex2.1 is the resulting tree. V→S put a copy of 'S' on the stack (Exercise #1) and V→M created a copy of 'S' in directory 'T' (Exercise #2). So now 'S' exists in both directory 'P' and directory 'T'. Put another way, a copy of directory 'S' has been "grafted" onto directory 'T'.

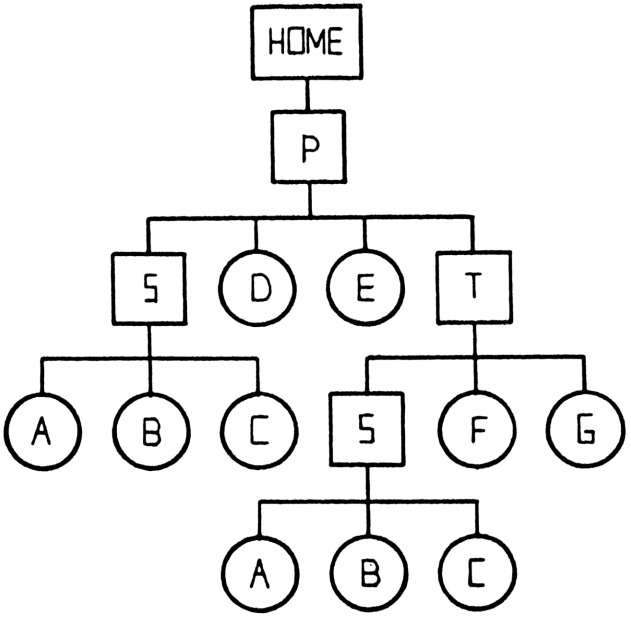


FIG Ex2.1: Result of Exercise #2.

What happens when you attempt to store a name that already exists? Using the STO (store) command, the old contents would be replaced by the new. For example, suppose variable 'K' contains the number 0.12; this means 0.12 is stored in a variable named 'K'. To change it from K=0.12 to K=0.35, one would simply type 0.35 'K STO. The variable 'K' would then contain the number 0.35. The new contents replaces the old as if STO tells the calculator, "Store this number under this name; if this name already exists then erase it and create this new one in its place."

V→M works a little differently. As a safety feature, V→M prevents the accidental over-writing (replacing or redefining) of variables and directories. V→M ends with the "Not Ready" message and does not alter memory if an attempt is made to over-write the contents of a name that already exits. V→M says "Store this variable if no other variable by the same name exists; otherwise stop and display a warning message." The next exercise demonstrates how the "Not Ready" condition occurs and what to do when it does occur.

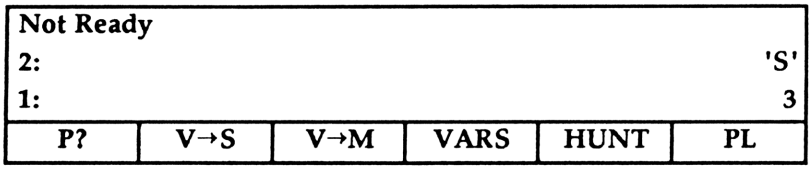
EXERCISE #3

Step	Type	Result
1.	PTRE ENTER	Restores directory 'P'.
2.	USER ℙ	Makes 'P' the current directory.
3.	« W , Z + ENTER 'B STO	Defines 'B' as a program. The program W + Z is stored in 'B'. The comma automatically changes to a space after pressing ENTER.

B	S	D	E	T	

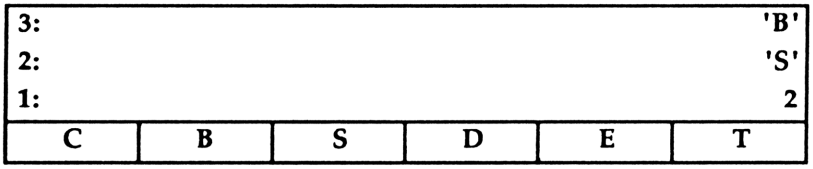
USER menu after step 3.

4.	𝕊	Goes to directory 'S'.
5.	CUSTOM VARS	Puts all the names of directory 'S' in a list on the stack ({ A B C }).
6.	𝕍→𝕊	Recalls all of those names.
7.	ℙℂ	Goes to the parent directory. The current path should be { HOME P }.
8.	𝕍→𝕄	Attempts to store 'A', 'B', and 'C' in directory 'P'.



The display after step 8.

9.	USER	Shows the USER menu (and clears the message flag).



The display after step 9.

The result is a new stack-coded directory that contains 2 names. FIG Ex3.1 and FIG Ex3.2 show what memory and the stack look like at this point. Originally there were 3 names in the stack-coded directory but 'C' was successfully created, leaving 'B' and 'A' on the stack. Why did it stop there? V→M attempted to create 'B' but stopped because 'B' already exists in the current directory. Executing V→M again will just result in the same "Not Ready" error, neither the stack nor the USER menu would change. What now? The existing 'B' must be purged before the new 'B' can be created. The exercise continues.

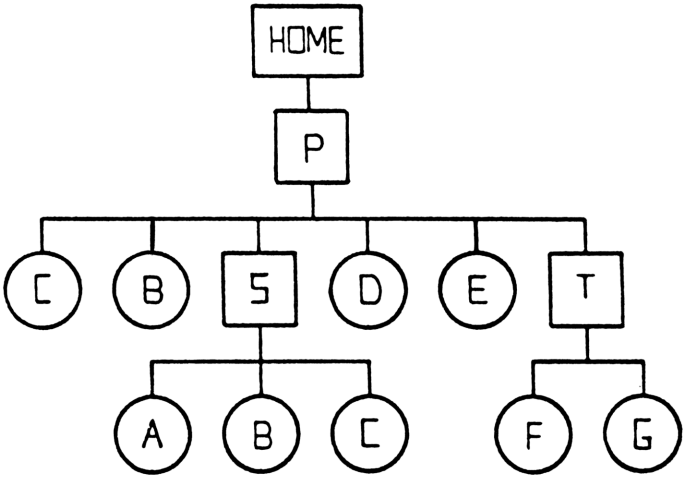


FIG Ex3.1: Memory Tree that results from Exercise #3.

```
6:    (2,1)
5:      'A'
4:    (2,2)
3:      'B'
2:      'S'
1:        2
```

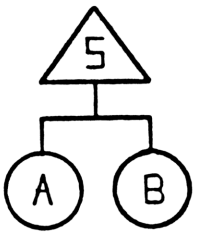


FIG Ex3.2: Stack-Coded Directory that results from Exercise #3.

EXERCISE #3 (continued)

Step	Type	Result
10.	'B RCL 'B2 STO 'B PURGE USER	Renames 'B' as 'B2'. The contents of 'B' are now in 'B2'.

B	C	S	D	E	T

USER menu before step 10.

B2	C	S	D	E	T

USER menu after step 10.

Step	Type	Result
11.	CUSTOM $\mathbb{V} \rightarrow \mathbb{M}$ USER	Resumes storing names.

A	B	B2	C	S	D

USER menu after step 11.

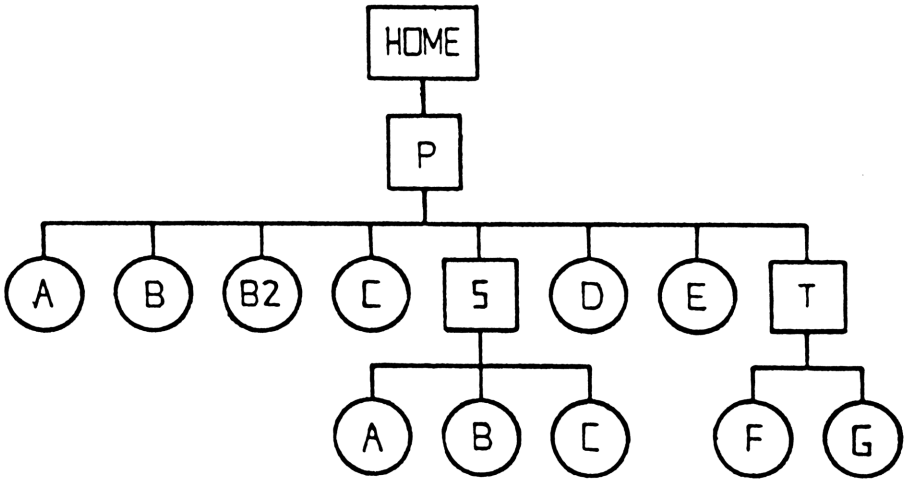


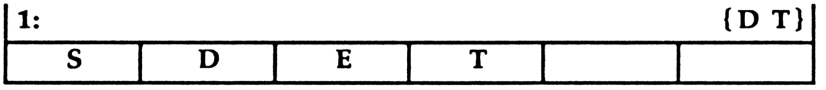
FIG Ex3.3: Memory Tree after step 11.

3.4 Purging Names

Typically, to purge a variable, one puts the name onto the stack and presses PURGE. But an attempt to purge a directory in that manner will result in a "Non-Empty Directory" error unless the directory is empty. Although this prevents the accidental erasure of a directory, it becomes an annoyance when you intend to erase non-empty directories. The next exercise illustrates how V→X purges both variables and directories alike, making it possible to free-up large amounts of memory quickly.

EXERCISE #4

Step	Type	Result
1.	PTRE ENTER	Restores 'P' Tree.
2.	USER ℙ	Goes to directory 'P'.
3.	{ D , T ENTER	Puts { D T } onto the stack. The comma automatically changes to a space and a closing brace is added after pressing ENTER.



The display after step 3.

4.	CUSTOM NEXT V→X	Purges.
5.	USER	Shows the USER menu. The menu labels show names S and E.

Variable 'D' and directory 'T' no longer appear in the USER menu because V→X erased them. FIG Ex4.1 shows the resulting tree.

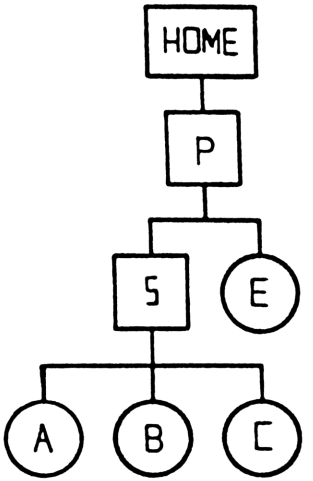


FIG Ex4.1: The result of Exercise #4.

3.5 Locating Names

Consider the tree of FIG 3.1. When 'P' is the current directory, 'G' does not appear in the USER menu. Suppose you do not know whether 'G' is in subdirectory 'S' or in subdirectory 'T' or in both 'S' and 'T'. How would you find 'G' without testing all possibilities? FIND reports the locations (directory paths) of a name as illustrated in Exercise #5.

EXERCISE #5

Step	Type	Result
1.	PTRE ENTER	Restores 'P' Tree.
2.	USER ℙ	Goes to directory 'P'.
3.	'G ENTER	Puts 'G' onto the stack. The menu labels show names S, D, E, and T.
4.	CUSTOM NEXT FIND	Executes FIND.

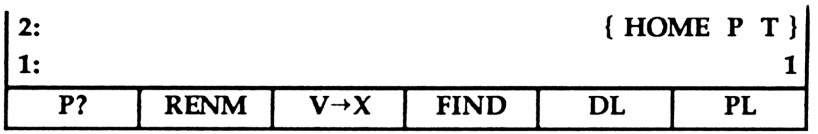


FIG Ex5.1: The display after step 4 of Exercise #5.

FIND reported the following: 1 copy of 'G' exists and it is located in the directory whose path is { HOME P T }. 'T' appears last in the path list and is said to be the parent of 'G'. ('T' is also the parent of 'F'; similarly 'P' is the parent of 'S', 'D', 'E', and 'T'.) For now, keep the results of Exercise #5 on the stack for use in the next exercise.

FIND reports locations but DL actually takes you to those locations as illustrated in Exercise #6.

EXERCISE #6

This exercise is to be performed immediately after exercise #5 so that

TREE = FIG 3.1
PATH = { HOME P }
STACK = FIG Ex5.1
MENU = CUSTOM

Step	Type	Result
1.	DROP	Removes the number from the stack.
2.	𝔻𝕃	Goes to the directory whose path matches that of stack level one ({ HOME P T } in this case).
3.	ℙ?	Helps you make sure you're in the right place. The path should be { HOME P T } at this point.
4.	USER	Shows the USER menu. The menu labels show names F and G.

3.6 Moving Names

Consider FIG Ex2.1. It is the result of performing Exercise #1 and Exercise #2 back-to-back. Compare FIG Ex2.1 to FIG 3.1. Notice that a copy of 'S' has been stored in 'T'. If the 'S' in 'P' were purged from FIG Ex2.1, then FIG 3.3 would be the result. Comparing FIG 3.3 to FIG 3.1 reveals that 'S' has been moved from directory 'P' to directory 'T'. This illustrates the steps involved in moving names from one directory to another. The steps are as follows.

A. Replicate: make a copy of the names.
   1. go to the directory that contains the names to be moved; that is called the Source Directory;
   2. put the names into a list; to move all names simply use VARS;
   3. use V→S to recall the names from the Source Directory;
   4. go to the final directory; that is called the Target Directory;
   5. use V→M to store the names into the Target Directory.

B. Clean Up: return to the Source Directory and purge the originals.
   1. return to the Source Directory, using PL (Locate Parent Directory) and DL (Directory Locate) as necessary;
   2. put the names into a list;
   3. use V→X to purge the names from the Source Directory.

Follow Steps A anytime you want to replicate a group of names. Follow both Steps A and Steps B anytime you want to move a group of names from one directory to another. You may even want to write a program that would do all of the steps automatically. Such a program would make excellent use of the FOX programs (DL, V→S, V→M, and V→X in particular) and would require three inputs: 1) the path of the Source Directory, 2) the path of the Target Directory, and 3) the list of the names to be moved.

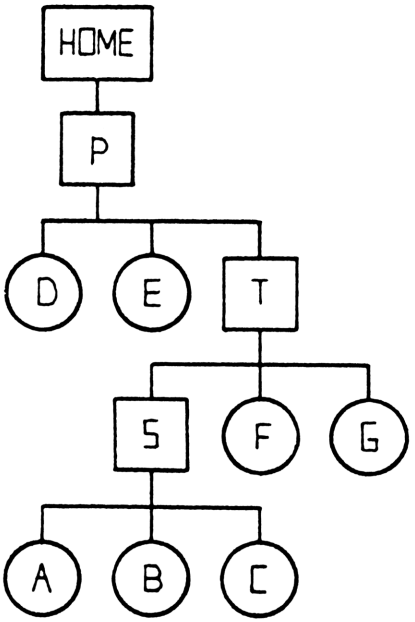


FIG 3.3: Result of modifying FIG Ex2.1.

3.7 Entering Programs

It takes approximately two hours to enter all 16 of the FOX programs. So before starting, use TABLE 4.1 (found on page 4-1) to select the programs you want to use, taking into consideration the operation, size, needs, and uses of each program. For the purpose of working the exercises of Section 3 select programs 1 through 12. Ideally you will eventually enter all 16 programs but if you are limited by available memory then you can use TABLE 4.1 to determine which programs to use.

Some of the programs require the existence of other programs in order to work properly. For each FOX program, TABLE 4.1 lists which other FOX programs are required to use it and which other programs depend on it. For example, V→M (program #9) needs three programs: #3, #4 and #5 (ALRM, PL, and DL). V→M is used by one program: #14 (RENM). So if V→M is not in memory then RENM will not work properly.

Before entering the programs, search through the HOME directory and all other directories to ensure that none of them contain names that match the FOX program names. Then, when entering the programs, stay in the HOME directory the whole time. All of the programs must be stored in HOME in order to work correctly.

If you are familiar with MENUS mode then you will be pleased to know that it is particularly suitable for entering FOX programs because all local variable names were selected from the lower half of the left keyboard. For even more convenience, set up the CUSTOM menu as follows: { V→S, V→M, V→X, PL, DL } MENU. That will make things easier.

After entering and storing a program, use the checksum program below. CKSM requires that stack level one contain the name of the program to be checked. CKSM returns a four-digit hexadecimal number. If the number does not match the one listed below for that program, then that program has not been entered correctly and/or has not been stored under the correct name.

```
« RCLF STD HEX 64
STWS 48 CF SWAP DUP
→STR SWAP RCL →STR +
16 STWS DUP # Øh 1
ROT SIZE
  FOR T OVER T DUP
SUB NUM R→B XOR RL
  NEXT →STR 3 6 SUB
ROT STOF SWAP DROP
»
'CKSM'
```

Checksum Program

[Please Note: this checksum program is the author's improvement version of a public domain program which is believed to have been posted on the HP public electronic bulletin board. The checksum program on this page is not a part of the FOX Project program set and is hereby regarded as public domain software. Responsibility for its use rests solely with the user.]

CHECKSUM VALUES FOR FOX PROGRAMS

1	MM	F699
2	MSET	E21F
3	ALRM	43FE
4	PL	654C
5	DL	E40B
6	P?	9F4E
7	PTRE	B7A3
8	V→S	7A5E

9	V→M	7D20
10	V→X	BFFB
11	FIND	1E7F
12	SEEK	A8AA
13	HUNT	3190
14	RENM	E0B4
15	CLDIR	EE48
16	ZAP	D9FA

SUMMARY OF PROGRAM ENTRY TIPS

- consider size and needs when selecting programs
- FOX names must be unique throughout all of USER memory
- all local variables are lower case and are found on the lower half of the left keyboard (letters s, t, v, w, x, y, and z; u is not used)
- CKSM verifies correct entry and correct naming

3.8 Onward

Having briefly read through this section once, you are ready to work through it hands-on. Enter the checksum program, store it as 'CKSM', and use it to verify that you have entered the checksum program correctly; the checksum for the checksum program itself is "5747" provided it has been stored as 'CKSM'. Next turn to Section 4, enter programs 1 through 12 into your calculator, using CKSM to verify each one. Then go back to the beginning of this section and work through the exercises.

PROGRAM LISTINGS

TABLE 4.1: FOX Project Program Size, Needs, and Uses.

#	NAME	OPERATION	SIZE*	NEEDS	USED BY
1	MM	Interface	26.5	2	.
2	MSET	Interface List	85.5	.	1
3	ALRM	Audio Prompt	65.5	.	9, 14, 15
4	PL	Locate Parent Directory	47	5	7-16
5	DL	Locate a Directory	41.5	.	4, 7 -16
6	P?	Identify Current Path	36.5	.	.
7	PTRE	Create 'P' Tree	307	4, 5, 10	.
8	V→S	Recall Names	219.5	4, 5	14
9	V→M	Store Names	332	3-5	14
10	V→X	Purge Names	139.5	4, 5	7, 14, 15
11	FIND	Locate a Name, Downwards	31	4, 5, 12	.
12	SEEK	Locate a Name, Downwards	179.5	4, 5	11
13	HUNT	Locate a Variable, Upwards	100	4, 5	16
14	RENM	Rename a Name	220	3-5, 8-10	.
15	CLDIR	Clear Current Directory	129	3-5, 10	.
16	ZAP	Purge a Name, Upwards	101	4, 5, 13	.

* Program sizes are approximate and are measured in kilobytes.

PROGRAM 1: MM

```
« MSET MENU
»
```

PROGRAM 2: MSET

```
{ P? V→S V→M VARS
HUNT PL P? RENM V→X
FIND DL PL }
```

PROGRAM 3: ALRM

```
« 900 .1 BEEP
900 .1 BEEP
»
```

PROGRAM 4: PL

```
« PATH DUP SIZE
1 - 1 MAX 1 SWAP
SUB DL
»
```

PROGRAM 5: DL

```
« DUP SIZE 1 1
ROT
      START GETI
EVAL
      NEXT DROP2
»
```

PROGRAM 6: P?

```
« PATH 1 DISP 1
WAIT CLMF
»
```

PROGRAM 7: PTRE

```
    « HOME { P }
V→X 'P' CRDIR P 'T'
CRDIR T (3,2) 'G'
STO (3,1) 'F' STO P
(1,2) 'E' STO (1,1)
'D' STO 'S' CRDIR S
(2,3) 'C' STO (2,2)
'B' STO (2,1) 'A'
STO HOME
    »
```

PROGRAM 8: V→S

```
    « 0 → v t
      « 31 CF
        IF v SIZE
        THEN 1 v
SIZE
          FOR x v x
GET
            IF VARS
2 PICK POS
            THEN
              IFERR
DUP RCL SWAP
              THEN
EVAL VARS V→S PL
              END
            ELSE
DROP t 1 + 't' STO
            END
          NEXT
        END PATH
DUP SIZE GET v SIZE
t -
      »
    »
```

PROGRAM 9: V→M

```
« 1 VARS → s t
x v
  « 30 CF
    WHILE x t ≤
30 FC? AND
    REPEAT
      IF DUP
TYPE
      THEN
        IF v 2
PICK POS
        THEN 30
SF
        ELSE
STO
```

[This program listing is continued on the very next column.]

[This is a continuation of the program listing for PROGRAM 9: V→M.]

```
        END
      ELSE
        IF v 3
PICK POS
        THEN 30
SF
        ELSE 2
PICK DUP CRDIR EVAL
V→M PL
        END
      END x 1 +
'x' STO
    END
    IF 30 FS?
    THEN s t x
- 2 + ALRM
"Not Ready" 1 DISP
    END
  »
»
```

PROGRAM 10: V→X

```
« → υ
  « 31 CF
    IF υ SIZE
    THEN 1 υ
SIZE
      FOR x υ x
GET
        IFERR
DUP PURGE DROP
        THEN
DUP EVAL VARS V→X
PL PURGE
        END
      NEXT
    END
  »
»
```

PROGRAM 11: FIND

```
« 0 SWAP SEEK
»
```

PROGRAM 12: SEEK

```
« VARS → z υ
  « 31 CF
    IF υ SIZE
    THEN 1 υ
SIZE
      FOR x υ x
GET DUP
        IFERR
RCL DROP2
        THEN
EVAL z SEEK PL
        END
      NEXT υ z
      IF POS
      THEN 1 +
PATH SWAP
      END
    END
  »
»
```

PROGRAM 13: HUNT

```
« PATH → z y
  « 0 1 PATH
SIZE
    START
      IF VARS z
POS
      THEN 1 +
PATH SWAP
      END PL
    NEXT y DL
  »
»
```

PROGRAM 14: RENM

```
« OVER 1 →LIST
→ y z v
  « 31 CF
    IF VARS y
POS VARS z POS NOT
AND
    THEN v V→S
DROP2 v V→X
      IF DUP
TYPE
      THEN DROP
z STO
      ELSE SWAP
DROP z SWAP 'x' 1
V→M
      END
    ELSE ALRM
"Not Ready" 1 DISP
    END
  »
»
```

PROGRAM 15: CLDIR

```
    «
      IF PATH SIZE
1 -
      THEN ALRM
PATH 2 DISP
"Delete?" 1 DISP
        DO
        UNTIL KEY
        END DUP 1
DISP
        IF "Y" SAME
        THEN VARS
V→X
        END CLMF
      END
    »
```

PROGRAM 16: ZAP

```
« PATH → z y
  « 1 PATH SIZE
    START
      IFERR z
PURGE
      THEN
      END PL
    NEXT y DL z
HUNT
  »
»
```

APPENDIX

A.1 Operation Requirements

ITEM	NOTE	RECOMMENDATION
User Memory	FOX Project programs require approximately 2061 bytes total. The amount of memory required for each program is reported in TABLE 4.1.	Have the appropriate amount of user memory available before entering in programs. Use the MEM command to check how much memory is available.
Modes	LAST mode is disabled by some operations.	Always expect LAST mode to be disabled after using any FOX program.
Flags	Some operations use User Flag 30. They do not require that it be set or cleared before hand but they may change it.	Always expect flag 30 to be affected by FOX programs.
Custom Menu	Program MM clears the CUSTOM menu.	A program like « { TAN RND IP } MENU » will conveniently reestablish the CUSTOM menu. If necessary, write and store such a program to quickly restore the CUSTOM menu. The names TAN, RND, and IP were arbitrarily chosen.

A.2 Nomenclature

- V→S means "Copy Names to the Stack."
- V→M means "Copy Names to the Current Directory" (into memory).
- V→X means "Move Names to the Abyss" (purge them).

A.3 Generalized Stack-Coded Directory

Program V→S puts a stack-coded directory onto the stack. Program V→M assumes the stack contains a stack-coded directory starting in level one. So the stack MUST be in the format shown in FIG A.2 prior to using V→M. Study FIG A.1 with FIG A.2. Notice that a stack-coded directory is simply a nested sequence of stack levels where each couplet is either a node or a cell. Each node consists of a directory name and the number of names contained in that directory. Each cell contains a variable and its name. The stack-coded directory in FIG A.2 was generated by using VARS and V→S from directory 'P' where the path was { HOME P }.

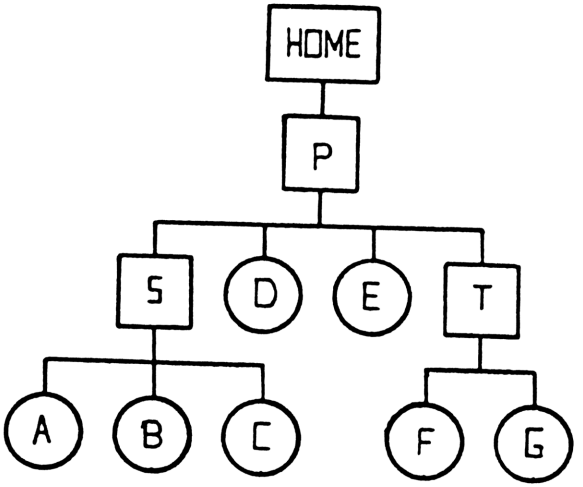


FIG A.1: The model tree used to get FIG A.2.

23:	any-thing	previous stack contents
22:	(2,1)	variable's contents
21:	'A'	variable's name
20:	(2,2)	variable's contents
19:	'B'	variable's name
18:	(2,3)	variable's contents
17:	'C'	variable's name
16:	'S'	directory's name
15:	3	# of names therein
14:	(1,1)	variable's contents
13:	'D'	variable's name
12:	(1,2)	variable's contents
11:	'E'	variable's name
10:	(3,1)	variable's contents
9:	'F'	variable's name
8:	(3,2)	variable's contents
7:	'G'	variable's name
6:	'T'	directory's name
5:	2	# of names therein
4:	'P'	directory's name
3:	4	# of names therein
2:	HOME	name of parent directory
1:	1	number of names recalled from the parent directory

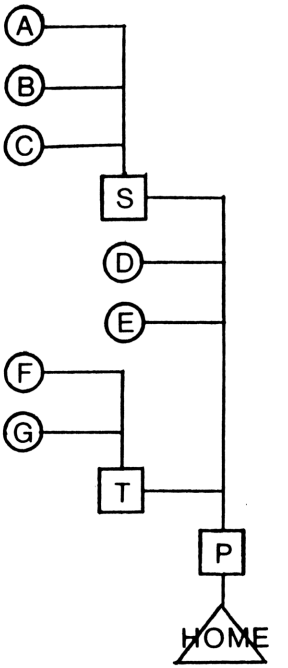


FIG A.2: (left) Stack-Coded Directory of FIG A.1; and (right) Generalized Stack-Coded Directory.