Little Universe: a Self-Referencing State Table

Karl-Dietrich Neubert
Physikalisch-Technische Bundesanstalt
Berlin, Germany

Abstract

Little Universe is a self-referencing state table, providing an environment in which to embed a multiplicity of alternative actions, procedures, equations or other objects. Essentially, it consists of matrices, whose rows are called classes and whose columns represent states. Each matrix element is considered to be an object, and a matrix is a world of objects. The set of all worlds is the universe. An object not only is identified by its class-, state-, and world-names, but conversely is aware of these coordinate values.

Introduction

Little Universe is a frame for implementing state tables for various kinds of applications. It is universal in a twofold sense: first, it may be used in structuring small knowledge bases or active systems with a high degree of flexibility; secondly, it provides a universal tool to implement certain coding structures with a uniform approach. We will first describe its logical structure and present the essential words; finally, we present and discuss some typical applications.

Structure

Little Universe consists of worlds, created by **WORLD=**. Each world is a matrix. The columns of this matrix are called states, and are created by **STATE=**. The world is set to any desired state simply by invoking the state name. The rows of the matrix are called classes, and are created by one of several class-defining words. Matrix elements can best be considered to be objects, because these elements are unrestricted with respect to type: they can be actions, functions, variables, constants, strings, or combinations of these. An object is activated by invoking its class name. The word **PROMPT** activates the object of the current class in the current world and state. Since rows and columns are identified by names and not by indices, the matrix can also be viewed as an associative memory.

The number of states has to be defined in advance by use of a word **STATES**. In contrast, the number of classes may be changed dynamically and is limited only by the resources at hand.

In a given vocabulary any name, i.e. world-, state-, and class-name, must be unique. We have implemented the ability of each object to communicate its membership with respect to class, state, and world. Furthermore, states and classes know from which world they descend. In physical memory, the entire matrix need not occupy contiguous memory space, but the objects in any single row, i.e., belonging to a given class, must.

The essential features of the underlying logical structure of Little Universe are shown in Fig. 1. At each level, some auxiliary cells are created to hold cross-references. In particular, each

world carries a number of cells for monitoring the total number of its states, the address of the current active state, the offsets of the last defined state and of the current active state relative to the first defined state, and finally a pointer to the last selected class name. A central role in this structure is played by the variable **ORG** (Origin) which holds the address of the current world. This information is also accessible from each state and each class. **ORG** plays the role of a mailbox and provides the path to environmental information with a minimum cost in memory cells. Fig. 1 also shows the sources of the available self-references, which are activated by the corresponding words **.CLASS**, **.STATE**, and **.WORLD**. Where appropriate, these self-references are activated automatically. By setting the Boolean value of the variable **DISPLAY?** to false, the screen display of these self-references may be suppressed.

To round out this general description, we have provided the source code for Little Universe as a text file in Appendix A, together with in-line comments. This version of Little Universe is written in PC/FORTH from Laboratory Microsystems, Inc. (LMI). In Little Universe, all words conform to the Forth-83 standard, except the still experimental words **BODY>** and **>NAME**, the word **PCKEY**, which is used in one example to read an extended keyset, the word **STRPCK**, which converts a string specified by address and length into a counted string, and some self explanatory words for screen attributes.

As examples of class defining words, the words **STRING=**, **VARIABLE=**, **CONSTANT=**, **FUNCTION=**, and **MATRIX=** are shown in Appendix A. Each of these defines a class of objects with common operational action. These and similar classes may be used in very different contexts. Class defining words that create more extended objects can also be envisaged, such as:

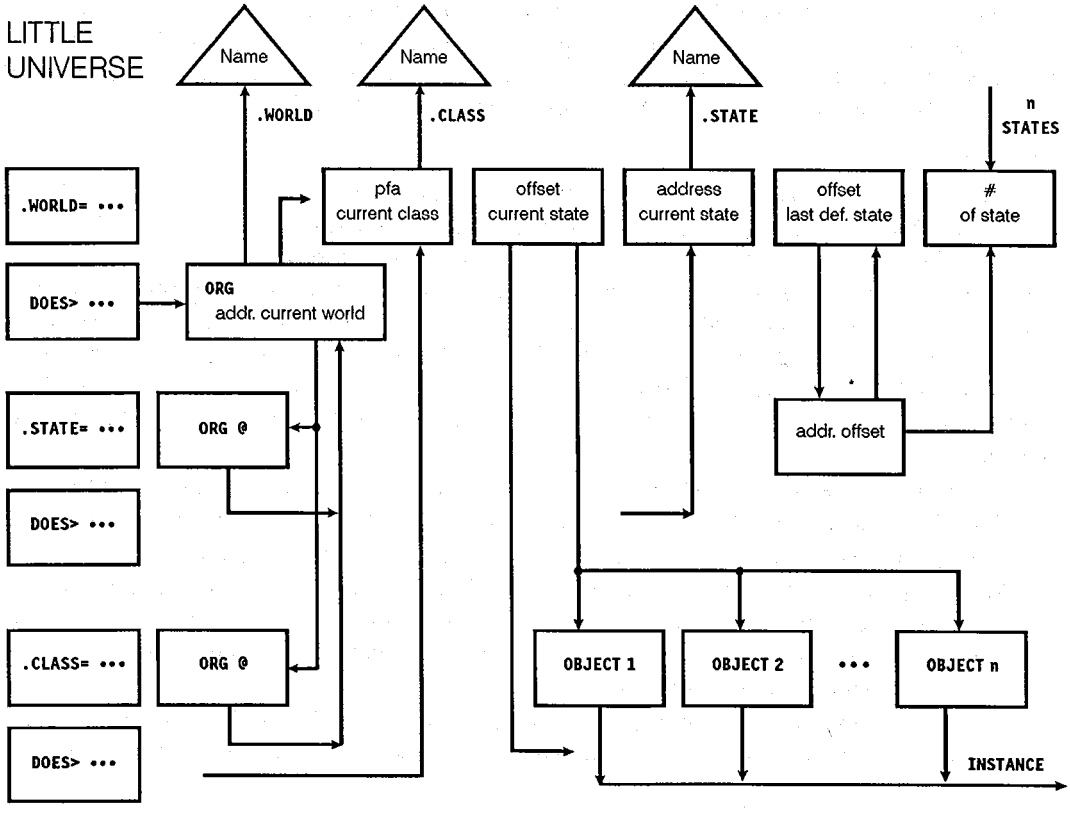

Fig. 1. Structure of Little Universe. Certain key information on constituent states is an integral part of each world, as seen in the uppermost boxes. The address of the current active world is stored in the variable **ORG**. A reference to **ORG** is part of each state and class object.

```
: COMPLEX=   SPECIFY COMMON PROPERTY   CREATE ...   DOES>   RECEIVE INFORMATION
   TRANSFORM INFORMATION   PERFORM ACTION   SEND INFORMATION ;
```

The flexibility of Forth allows one to implement almost any conceivable run-time action of an object.

As an overview of Little Universe, its user vocabulary is listed in Appendix B, with accompanying notes on usage.

Sample Applications

Executable code is provided in Appendix C to illustrate various applications. To use Little Universe, one must first define a world. Then the world must be activated and the number of states in this world must be specified. At least one state must be defined and activated before a class may be defined. Further states can then be defined. These different actions may be intercepted by any other Forth words. In particular, one may alternate between different worlds during a session. The correct world is set automatically, if not specifically set by the user. Similarly, objects of different worlds can be activated by the user without providing the world specification.

The first two examples below are intended to demonstrate the basic approach for implementing a small knowledge base or a small active system. The remaining examples illustrate that various standard code structures may be viewed under one common aspect, namely as specific applications of Little Universe. Furthermore, to make the vocabulary more relevant, we introduce in the examples appropriate synonyms in place of the standard wordset of Little Universe. Obviously, the introduction of such synonyms is not essential to the execution of the code.

Example 1: a knowledge base

This example represents a knowledge base that provides information on two planets in a world named **PLANETS**. In addition to the classes **COLOR** and **#SATELLITES**, arbitrarily many other classes with various properties may be defined. With the given definitions, the user will be rewarded with the requested information by typing **EARTH COLOR** or **MARS COLOR**.

Even though with Little Universe we have only one matrix at our disposal for each world, extensive systems may be built. Consider an universe of say four worlds with 40 states and 40 classes each, then there are already 6,400 objects at hand, some of which may be by themselves matrices of arbitrary size. Nevertheless, only 320 words would be added to the Forth vocabulary! By using different vocabularies, several such universes may be defined concurrently. One may view each vocabulary, each world, and each state as a node in a hierarchical structure, which then has four levels, when classes are also considered as a level. With some loss of simplicity, Little Universe may be extended to model hierarchical structures of still greater depth. For example, to represent different worlds as different states of a class **WORLDS=** would require the generalization of the variable **ORG** to a vector, to hold state information of additional levels.

Example 2: an active system

This example differs from the first example in the type of classes used. Here the classes are functions. In addition, these functions accept control parameters at runtime. These functions are intended to represent particular real-time action performed by some machinery. In the world **ELEVATOR**, the command **6 UP GO** activates an elevator to go up 6 units, **5 DOWN GO** to go down 5 units. This example represents a wide range of possible applications. In medical instrumentation, it could represent a heart-lung machine interpreting instructions differently under different patient conditions. In the field of robotics, it could represent a robot behaving differently in different environments. Furthermore, if a class **CLASS= SENSE-ENVIRONMENT SET-STATE** is implemented, the robot will be able to monitor the environmental condition continuously and automatically set the appropriate environmental state.

Example 3: deferred definitions

In Little Universe a class created by **FUNCTION=** may be referred to before it is initiated. We take advantage of this fact in the case of deferred definitions. We use the class **FUNCTION=** under the synonym **DEFER** and initiate the function using the synonym **RESOLVE** for **COIN**. Since the number of rows in the state table is unlimited, arbitrarily many deferred definitions may be handled this way.

Example 4: vectorization

Vectorization is a preferred method in Forth when one wants to use identical code in changing situations or for different purposes. In the given example, **TELL** will print different sentences depending on whether the current active vector component is **VEC1**, **VEC2** or **VEC3**.

Example 5: the case construct

The case construct may be implemented in two ways, either as an ... **IF** ... **THEN** structure or by use of a state table. In the first way, the conditional branch to be performed depends on the numeric value of a variable. In Little Universe, the action to be taken is the direct execution of a state-dependent named operation. This allows a straightforward solution to an otherwise complex situation.

As an illustration, consider the problem of designing a hot-key mode for the keyboard. We want the response action to depend on the pressing of two keys: the first key should be any key from F1 to F10, the second key any capital letter. This lets the computer user choose any one of 260 different actions quickly with only two keystrokes; such efficiency may be required in a critical situation. The Little Universe solution is given in example 5. First, we define a word **HOT** which initiates a key request which runs indefinitely until the ESC key (ASCII code 27) is detected. Further, it maps by a constant shift the numeric codes of keys F1 to F10 onto the numeric codes of the letters a, b, c, ... , and interprets these codes as strings with the value a, b, c, ... , so that they can be searched for in the Forth dictionary. Next, in a world **CASES**, we define under the synonym **CASE** the states a, b, c, ... , and assign to each capital letter for each **CASE** the required action. In example 5, this is shown for the letter **Q** and **W**. These few steps are all that must be done to make **HOT** to perform according to the set of specifications: if F1 has been pressed, **Q** takes one particular action; if F2 has been pressed, **Q** performs some other action; and so on. Compare this solution to a solution of the form **ASCII F1 = SWAP ASCII Q = AND IF ... THEN** for large sets of hot-key combinations!

Note that in the frame of object-oriented programming the problem of transforming numerical values to callable string names may be quite general. It may not always be easy to find a mapping which is straightforward and at the same time produces meaningful names to use as case names.

Example 6: local variables

The quest for local variables seems to have its source in the convenience of using variable names from the "main" program in callable subroutines or words as well, without clashes. A further justification of local variables is to avoid cluttering the dictionary with names which are only of local usage. In the world **LOCALS**, the number of states is the anticipated number of Forth words where local variables may be needed. Then, using the synonym **LOCAL** for **VARIABLE=**, we may define arbitrarily many local variables. For example, if one has **2Ø LOCAL** uses occurring in 30 Forth words, **W1**, **W2**, **W3**, through **W3Ø**, then one can use a total of 600 independent local variables at the cost of only 50 dictionary entries. This method is also very memory-efficient, in that **VARIABLE=** creates a constant overhead independent of the number of states.

In strict usage, the value of a local variable is allowed to be lost once the subroutine is exited. If one intends to use this kind of local variable, then two states are sufficient: one for the "main" program and the other for the subroutines. It might be attractive to coin names for the two states such that the syntax reflects the particular locality of the variables. In the example, **STATE=}** refers to the main program, **STATE={** refers to the subroutines.

Summary

Little Universe is an environment for creating state tables in a wide range of applications. It is easily extended to cover the needs of special applications. These tables are indeed state tables in the technical sense used by Brodie **[1]**, but have a greater functionality. Further, they exhibit self-referential features, as described by Smith **[2]**, in that they refer to their own internal state and their location in the universe. Some of the examples given provide alternative solutions to problems discussed in recent issues of *Forth Dimensions*.

Acknowledgment

Special thanks to my colleague Christopher McManus, who recast the phrasing of this paper to increase its readability. His discussions over the paper's substance has also improved its content.

References

[1] Leo Brodie, *Thinking Forth, a Language and Philosophy for Solving Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1984. P. 219.

[2] B.C. Smith, "Self-Reference," *Encyclopedia of Artificial Intelligence*, editor Stuart C. Shapiro. John Wiley and Sons, New York, 1987. Vol. 2.

Dr. Karl D. Neubert received a M.Sc. in Physics from Case Western Reserve University in Cleveland, Ohio in 1960 and a Ph.D. in Physics from the Technical University of Berlin, West Germany in 1969. He has spent several years in solid state research, especially studying the mechanical and electrical properties of silicon, and presently leads a laboratory at the Physikalisch-Technische Bundesanstalt in Berlin where they are developing a body function simulator for testing medical instrumentation. His other interests include parallel computing and expert systems, for which Forth is a promising alternative to other tools.

APPENDIX A : Source Code

```
: TITLE CLS
  15 5 GOTOXY ."            Little Universe:                "
  15 7 GOTOXY ." a Self-Referential State-Table Environment "
  15 9 GOTOXY ."       Copyright 1988, K.D.Neubert          " ;
  TITLE
( -------------------- VARIABLES -------------------------- )

  CREATE ORG Ø ,              ( reserved for world address )
  CREATE EXECUTE? 1 ,        ( 1: execute, Ø: specify function )
  CREATE DISPLAY? 1 ,        ( 1: display, Ø: suppress display )

( ------------------- DIAGNOSTIC WORDS -------------------- )

: NAME ( pfa --- type < name >)            ( pfa --> type name )
     BODY> >NAME COUNT
     63 AND                               ( mask off the 3 MSB )
     SPACE TYPE SPACE ;
: .WORLD ( --- )
     CR ." current WORLD is " ORG @
     REVERSE NAME REVERSE SPACE ;
:  STATE? ( --- )
     ORG @ 4 + @ Ø= IF ABORT" no STATE activated " THEN ;
: .STATE ( --- )
     STATE?
     CR ." current STATE is " ORG @ 4 + @
     INTENSITY NAME -INTENSITY
     ?XY SWAP DROP 32 SWAP GOTOXY ;
: .#STATES  ( --- )        ( how many states are not defined ? )
    .WORLD ." with " ORG @ 2+ @ 2/ . ." of " ORG @ 6 + @ .
    ." STATES still to be defined  " ;
: STATES?  ( --- )                  ( is # of states specified ? )
     DISPLAY? @ IF
       .WORLD
       ORG @ 6 + @ Ø= IF
       INTENSITY ." assign n STATES , n > Ø ! " -INTENSITY
       THEN
     THEN ;
: .CLASS ( --- )
     ORG @ 8 + @ Ø= IF ABORT" no CLASS activated " THEN
     CR ." current CLASS is " ORG @ 8 + @
     INTENSITY NAME -INTENSITY SPACE ;
```

```
: CLASS! ( pfa --- pfa )
     DUP ORG @ 8 + ! ;          ( keep pfa for later reference )
: WORLD?!  ( pfa --- pfa )
     DUP @ DUP ORG @ <> IF
        ORG !                              ( set current world )
        DISPLAY? @ IF .WORLD THEN
     ELSE DROP THEN ;
: COORDINATES ( pfa --- pfa )
     WORLD?!
     DISPLAY? @ IF .STATE THEN
     CLASS! ;
( -------------------- STRUCTURING WORDS -------------------- )

: WORLD=   ( name )           ( arbitrarily many definitions  )
     CREATE Ø ,               ( offset of last selected STATE )
            Ø ,               ( offset of last defined  STATE )
            Ø ,               ( address of last selected STATE)
            Ø ,               ( total # of STATES             )
            Ø ,               ( pfa of last selected CLASS    )
     DISPLAY? @ IF .WORLD THEN   ( reminder to activate world )
     DOES>  ( --- )
        ORG !
        STATES? ;
: STATES ( # of --- )   ( assign # of STATES in current WORLD )
     ORG @ 6 + @ Ø> IF ABORT" already assigned ! " THEN
     DUP 2* ORG @ 2+ ! ORG @ 6 + ! ;
: STATE=  ( name )                  ( # of STATES definitions )
     ORG @ 2+ @ 1 < IF ABORT" definitions exhausted ! " THEN
     -2 ORG @ 2+ +!
     CREATE ORG @ , ORG @ 2+ @ ,       ( address of world-name,)
                                       ( offset with respect to)
     DOES> ( --- )
        WORLD?!
        DUP ORG @ 4 + !
        2+ @ 2+ ORG @ ! ;

WORLD= VAST VAST TITLE          ( default setting for world )
( ----------------- OBJECT ACTIVATING WORD ----------------- )

: PROMPT ( --- )                              ( activate object )
     DISPLAY? @ IF .CLASS THEN
     ORG @ 8 + @ BODY> >NAME COUNT 63 AND
     STRPCK FIND DROP EXECUTE ;
```

```
( ------------------ CLASS CREATING WORDS ------------------- )

: VARIABLE= ( name )                          ( create variable )
     STATE?
     CREATE ORG @ , ORG @ 6 + @ Ø DO Ø , LOOP
     DOES>
        COORDINATES
        DUP @ @ + ;       ( -- addr )

: FUNCTION=  ( name )                         ( create function )
     STATE?
     CREATE ORG @ , ORG @ 6 + @ Ø DO ['] NOOP , LOOP
     DOES>
        COORDINATES
        DUP @ @ +
            EXECUTE? @ IF @ EXECUTE          ( execute function )
                ELSE !                       ( specify function )
                     1 EXECUTE? !
            THEN ;
: COIN      ( COIN < predefined word name > < function name > )
     Ø EXECUTE? !
     ' ;
: CONSTANT= ( name )                          ( create constant )
     STATE?
     CREATE ORG @ , ORG @ 6 + @ 2* 1- Ø DO Ø , LOOP
                ( set fields for constants and marks to zero )
     DOES>
        COORDINATES
        DUP @ @ + DUP @
        IF @                  ( -- n )          ( get constant )
        ELSE
           DUP ORG @ 6 + @ 2* + DUP @     ( inspect mark field )
           IF DROP @           ( -- n )      ( constant is Ø ! )
              ELSE 1 SWAP !                   ( set mark field )
                           !   ( n -- )        ( set constant )
           THEN
        THEN ;

: STRING= ( name )                    ( create string constant )
     STATE?
     CREATE ORG @ , ORG @ 6 + @ Ø DO Ø , LOOP
     DOES>
        COORDINATES
        DUP @ @ + DUP @ IF @                    ( string addr )
                  COUNT TYPE                    ( type string )
        ELSE
           HERE SWAP !                    ( set field to HERE )
           DUP HERE OVER C@ 1+ CMOVE     ( move string to HERE )
           C@ 1+ ALLOT                           ( update HERE )
        THEN ;
```

```
                : BOUNDS? ( n n pfa --- )  ( used in MATRIX= )
                          ( to check upper bounds of indices )
                     >R 2DUP
                     R@ 2+ @ 1- > SWAP R@ 2+ 2+ @ 1- > OR IF
                       ." index overflow : "
                       R> 2+ 2@ . ." x " . ." matrix ! " ABORT
                     ELSE R> THEN ;
: MATRIX= ( n1 n2 .. name              ( create n1 x n2 matrix )
     STATE?
     CREATE ORG @ , 2DUP , ,
     ORG @ 6 + @ Ø DO 2DUP * 1+ I * ORG @ 6 + @ + 3 + 2* , LOOP
                              ( produce pointers to matrices )
     ORG @ 6 + @ Ø DO 2DUP DUP , * Ø DO Ø , LOOP LOOP 2DROP
                                 ( set matrix elements to Ø )
     DOES>  ( #y #x -- addr ) ( indices from Ø,Ø to n1-1,n2-1 )
       COORDINATES
       BOUNDS?                                  ( check indices )
       DUP DUP @ @ + 4 + @ +       ( addr of first matrix elem.)
       DUP @ ROT * ROT + 2* + 2+ ; ( addr of #y,#x matrix elem)
```

APPENDIX B : User Vocabulary

```
(                    STRUCTURING WORDS

              WORLD= < name >     define world
                     < name >     activate world
            n STATES              assign # of STATES
              STATE= < name >     def. state in current world
                     < name >     activate state

                    OBJECT ACTIVATING WORD

         PROMPT      activate objekt  of current class
                     in current state of current world
)
(                  CLASS DEFINING WORDS

           VARIABLE= < name >     define variable
                   n < name > !   assign n to variable
                     < name > @   get value of variable

           FUNCTION= < name >     define function
COIN ,< predef. name > < name >     specify function
                     < name >     execute function

           CONSTANT= < name >     define constant
                   n < name >     assign n to constant
                     < name >     place constant on stack

             STRING= < name >     define string constant
             " xxx " < name >     assign xxx to string
                     < name >     display string

       n1 n2 MATRIX= < name >     define n1 x n2 matrix
             n #y #x < name > !   assign n to element #y #x
               #y #x < name > @   get value of #y #x element
                                  #y,#x = Ø,Ø to n1-1,n2-1
)
(                    DIAGNOSTIC WORDS

         .WORLD      display name of current world
         .STATE      display name of current state
         .CLASS      display name of current class
         .#STATES    display the # of states still
                     to be defined in current world

       1 DISPLAY? !  automatic display of diagnostics on  !
       Ø DISPLAY? !  automatic display of diagnostics off !
)
```

APPENDIX C : Examples

```
    Ø 23 GOTOXY .( press any key to load  EXAMPLES ) KEY DROP
    Ø DISPLAY? !

( ---------------------- Example 1 ------------------------ )

WORLD= PLANETS PLANETS                                  9 STATES
STATE= MERCURY STATE= VENUS STATE= EARTH STATE= MARS ( etc.)
MARS                        STRING= COLOR  CONSTANT= #SATELLITES
      " This is the blue planet.  " COLOR          2 #SATELLITES
EARTH " This is the green planet. " COLOR          1 #SATELLITES


( ---------------------- Example 2 ------------------------ )

: up    ( n --- )     Ø DO       I . LOOP      ;
: down  ( n --- ) DUP Ø DO DUP I - . LOOP DROP ;
: alarm ( n --- )                 1ØØ * 2Ø BEEP ;

WORLD= ELEVATOR  ELEVATOR          3 STATES
STATE= UP     STATE= DOWN     STATE= ALARM
UP    FUNCTION=  GO       VARIABLE=  SPEED
      COIN up    GO                  5 SPEED !
DOWN  COIN down  GO                 1Ø SPEED !
ALARM COIN alarm GO                  Ø SPEED !

( ---------------------- Example 3 ------------------------ )

: DEFER FUNCTION= ;
: RESOLVE COIN ;

WORLD= ANY                     ANY 1Ø STATES
STATE= ANY-ONE
ANY-ONE    DEFER INSIGHT
                 INSIGHT                ( no action performed )
: SOLUTION ."    This was evident. " ;
RESOLVE SOLUTION INSIGHT
                 INSIGHT               ( SOLUTION is executed )

( ---------------------- Example 4 ------------------------ )

: ESTIMATE ." Every activity takes more time than you have. " ;
: SIMPLIFY ." Nothing is ever as simple as it first seems.  " ;
: CLARIFY  ." Every clarification breeds new questions.     " ;

: VECTOR FUNCTION= ;

WORLD= VECTORIZATION       VECTORIZATION 3 STATES
STATE= VEC1    STATE= VEC2   STATE= VEC3
VEC1        VECTOR TELL
     COIN ESTIMATE TELL
VEC2 COIN SIMPLIFY TELL
VEC3 COIN CLARIFY  TELL
```

```
( ---------------------- Example 5 ------------------------ )

: HOT ( --- )                          ( initiate hot key mode )
  BEGIN
      PCKEY ?DUP Ø= IF                ( request key, special key ? )
        38 + THEN                     ( map F1...F1Ø.. onto a...j.. )
      DUP 27 = IF
        DROP QUIT THEN                ( quit hot key mode on ESCAPE )
      PAD ! PAD 1 STRPCK                 ( prepare string for FIND )
      FIND IF EXECUTE                              ( execute string )
           ELSE ." not defined " DROP THEN
      ?XY 24 = SWAP 64 > AND IF CLS THEN
  AGAIN ;

: case STATE= ;

WORLD= CASES                CASES 1Ø STATES
case a case b case c case d ( etc )
a  STRING= Q STRING= W        ( etc )
   " Who  are you ?  " Q        b  " What are you ?    " Q
c  " Why  are you ?  " Q        d  " How  are you ?    " Q
a  " Wer bist Du ?   " W        b  " Was bist Du ?     " W
c  " Warum bist Du ? " W        d  " Wie geht es Dir ? " W

( ------------------------ Example 6 ---------------------- )

: LOCAL VARIABLE= ;

WORLD= LOCALS  LOCALS                 3Ø STATES
   STATE= W1  STATE= W2  STATE= W3    ( etc.)
W1 LOCAL xi   LOCAL xj    LOCAL xk

   STATE= }  STATE= {      ( define states "main","subroutine" )
}                                ( set "main" as default state )
: MAIN-1  42 xi ! ;
                  : SUB {             ( set "subroutine" state )
                        24 xi !
                      } ;              ( reset to "main" state )
: MAIN-2  xi @ . ;
MAIN-1  SUB  MAIN-2                                ( prints 42 )
```

```
( ————————- RUN some EXAMPLES ————————— )

1 DISPLAY? !
CR .( press any key to run some of the EXAMPLES ) KEY DROP CLS

MARS  #SATELLITES .CLASS   .          COLOR
UP          SPEED .CLASS @ .          12 GO
PLANETS    PROMPT .#STATES      DOWN   6 GO
TELL    VEC2 TELL               ALARM  2Ø GO         UP  VEC1

CR CR  .(      Try yourself: )
CR .(  enter   HOT enter     F1 Q W    F4 Q W      Esc       )
CR .(  enter   COLOR  12 GO  EARTH PROMPT  TELL  ANY PROMPT   )
CR .(  enter          ØØDISPLAY? !         and try examples  )

( ———————————————————————— )
```