DOCUMENT RESUME

ED 323 960 IR 014 598

AUTHOR	Avner, Elaine
TITLE	Summary of the uTutor Language. Second Edition.
INSTITUTION	Illinois Univ., Urbana. Computer-Based Education Research Lab.
PUB DATE	Oct 89
NOTE	125p.; The language is referred to throughout as "uTutor" with the "u" being the Greek letter "Mu". For additional information, see ED 124 149.
PUB TYPE	Guides - General (050) -- Computer Programs (101)
EDRS PRICE	MF01/PC05 Plus Postage.
DESCRIPTORS	*Authoring Aids (Programing); *Programed Tutoring; *Programing Languages
IDENTIFIERS	*TUTOR Programing Language

ABSTRACT

This summary presents features of the uTUTOR programming language. Intended for the experienced author who needs a quick reference for the form of a tag and for some of the restrictions on commands, it does not discuss fine details of the uTUTOR language. Topics covered are: (1) calculating, including operations and symbols, system functions, random numbers, and system variables; (2) file operations, including datasets, name sets, and directories; (3) judging, including preparation for responding, modification of and matching the response, and alteration of judgment and feedback; (4) presenting, including basic display, graphics, color display, and non-screen presentation; and (5) sequencing, including automatic and key-initiated sequencing, pausing and timing, lesson connections and section, and signing off. Appendices display: (1) keysets, character codes, hexadecimal numbers, and powers of two; (2) alphabetical indexes to system variables and commands. (DB)

I.R.

Computer-based Education Research Laboratory

CERL

SUMMARY OF THE μTUTOR LANGUAGE

ELAINE AVNER

SECOND EDITION

OCTOBER 1989

University of Illinois at Urbana-Champaign

SUMMARY OF THE μTUTOR LANGUAGE

Elaine Avner

Computer-based Education Research Laboratory

University of Illinois at Urbana-Champaign

First Edition July 1984
Second Edition October 1989

Acknowledgment

Members of the Systems Staff of the Computer-based Education Research Laboratory at the University of Illinois developed the µTUTOR language. Several members of the Systems Staff and the User Services staff made valuable comments on the first edition and the current edition of this summary of the language.

Bruce Sherwood made many suggestions about the content of this book.

Sam Milosevich explained details of color features. He also discussed the meaning of "stand alone" in the Cluster environment.

Charles Bridges reviewed information on file operations.

Judith Sherwood suggested several changes and corrections to the first edition. She raised questions about some documented features that resulted in clarification of descriptions.

The keyset diagrams in the appendix are based on diagrams designed by Judith Sherwood.

Wayne Wilson assisted with final preparation of the manuscript.

This summary presents features of the µTUTOR programming language. It is intended for the experienced author who needs a quick reference for the form of a tag and for some of the restrictions on commands. It does not discuss fine details of the µTUTOR language. For such information authors should refer to *The µTUTOR Language* by Bruce Sherwood and Judith Sherwood.

Each command includes a brief description of its purpose and a description of the tag. The standard form is

```
command  brief description of operation of command

         command DESCRIPTION OF TAG   (any explanatory comments)

         Note:  Additional comments about this command.

NOTE:  General comments about groups of commands.
```

In the description of a tag, words in upper case represent variables, values, expressions, or character strings supplied by the author; words in lower case are required words in the tag and must appear exactly as shown. Arguments in the tag which are optional are printed in italics. Optional arguments can be required words or author-supplied information.

For example:

```
at      FINEX,FINEY
draw    LOCATION1;skip;LOCATION2
lesson  complete
ansv    EXPR,TOLERANCE
```

The commands are grouped into six categories:

calculating (C)
file operations (F)
judging (J)
presenting (P)
routing (R)
sequencing (S)

Commands which are difficult to classify are placed in categories which describe their most probable use.

CONTENTS

C

F

J

P

R

S

A

I

Abbreviations and Notes

These abbreviations are used in the descriptions of command tags:

abbreviation	definition
arg	argument or tag entry
(b)	blank tag
char	character
coarse	coarse-grid coordinates
disk	magnetic storage disk
expr	mathematical expression
finex,finey	fine-grid coordinates
num	number of
string	character string
var	variable

In conditional statements and in statements where a variable is set, suffixes M, 0, 1, 2, etc., denote the minus condition, 0 condition, 1 condition, 2 condition, etc., e.g.,

```
keytype VAR,KEY0,KEY1,KEY2
do      EXPR,NAMEM,NAME0,NAME1,NAME2
```

In conditional statements the conditional expression is <u>rounded</u> (not truncated) to the nearest integer. Thus, a value of -.4 results in the 0 condition being selected rather than the minus condition.

Generally, wherever a tag entry can be a number, a mathematical expression is also accepted.

Command names are enclosed in dashes when they are referred to in descriptions, e.g., -next-. Names of system variables are enclosed in double quotes, e.g., "zreturn". Key names are capitalized, e.g., NEXT. A function key name followed by "1", e.g., NEXT1, indicates the SHIFT key is held while the key is pressed.

Commands labeled "non-executable" are active only when the lesson is being condensed and not during execution.

When variables are used in the tag of certain commands which require names in the tag, e.g., -jumpout-, the variable must be enclosed in parentheses to indicate that the information needed is the contents of the variable and not a character string; e.g., -jumpout (var)- means the file whose name <u>is contained</u> in a buffer starting with variable var, while -jumpout var- means the file whose name <u>is</u> var.

CALCULATING

C

Basic calculating C1

define
calc
calcc
calcs
zero
set
compute
block

Operations and symbols C4

+ - x * ÷ / $divr$ $divt$ **
( ) [ ] { } ⇐
¢ ° 16# 2# 8# ""

System functions C5

abs frac int sqrt
alog log exp ln
sin cos tan cot sec csc
arcsin arccos arctan
arccot arcsec arccsc
zk zlength zvloc
= ≠ ≤ ≥ < >
and or not
rsh lsh ars cls
$mask$ $union$ $diff$
comp bitcnt

Random numbers C7

randu
setperm
randp
remove
restore

Character manipulation C8

pack
packc
search
searchf

Operations on lists C10

find

System variables for computing C11

zretinf

FILE OPERATIONS

Datasets F1	Directories F7
setfile	setfile
addfile	addfile
delfile	delfile
chgfile	chgfile
getfile	getfile
datain	names
dataout	setdir
reserve	reserve
release	release

Namesets F3	System variables for file operations F9
setfile	zfauth(FIP)
addfile	zfbpi(FIP)
delfile	zfbpn(FIP)
chgfile	zfbpr(FIP)
getfile	zfmaxn(FIP)
names	zfnams(FIP)
setname	zfrecs(FIP)
addname	zftype(FIP)
delname	zretinf
chgname	zreturn
getname	
addrecs	
delrecs	
datain	
dataout	
reserve	
release	

JUDGING

J

PRESENTING

ROUTING

Router lesson and curriculum information R1

router
lesson
score
status

System variables for routing R2

zldone
zrouten
zscore
zstatl

R

SEQUENCING

S

C

CALCULATING

Basic calculating

define (non-executable) defines names of variables, constants, arrays, and functions; type is 16-bit signed integer (i,16:) unless specified as floating point (f,48: or f:), 8-bit signed integer (i,8:), or 8-bit unsigned integer (b:); all definitions following a specified type have that type until a different type designation is encountered

for example:

```
define  NAME1,NAME2,NAME3
        NAME4(ARRAYSIZE)
        i,8:NAME5,NAME6
        b:NAME7
        FUNC(ARG1,ARG2)=EXPR
        f,48:NAME8,NAME9(ARRAYSIZE)
        i,16:NAME10
        i,8:NAME11≡NAME9,NAME12≡NAME9
        NAME13=2001,NAME14=4.3
```

Note: Defined names can contain up to 7 characters and must start with a letter.
Up to 6 arguments are permitted in defined functions.
One-dimensional arrays are permitted.
Approximately 1000 definitions are permitted.
Variables are allocated in memory in the order in which they are defined. Constants and functions are not stored in memory.
The define set must be placed before the first -unit- command.

A local define set is declared as a continuation of a -unit- command. (The -define- command is omitted.) Formats for local variables are the same as those for global variables.

To merge the local define set with the global define set:

```
unit    someu
        merge,global:   (final colon is required)
        a,b,d
        f:result
```

examples of definitions:

```
define  a,b,c          $$ 16-bit integers
        f:radius       $$ floating point
        b:char(10)     $$ 8-bit unsigned array of 10 elements
        sinh(x)=.5x(exp(x)-exp(-x))  $$ one-argument function;
        cosh(x)=.5x(exp(x)+exp(-x))  $$ x cannot be previously defined
        average(sum,N)=sum/N         $$ two-argument function
        stdev(sum,sumsq,N)=sqrt((Nxsumsq-sum**2)/(Nx(N-1)))
```

calc assigns the value of the expression on the right side of the assignment arrow to the variable on the left side (operations and functions are given at the end of this subsection on Basic calculating)

```
calc    VAR←EXPR
calc    VAR←"CHARACTER"   (single character only; character code is
           placed in the right-most 8 bits of VAR of integer type)
```

calcc performs one of several calculations depending on the rounded value of a conditional expression

```
calcc   EXPR,VAR1←EXPRM,VAR2←EXPR0,VAR3←EXPR1,,VAR4←EXPR3
```

calcs sets a variable to one of several values depending on the rounded value of a conditional expression

```
calcs   EXPR,VAR←EXPRM,EXPR0,EXPR1,EXPR2,,EXPR4
```

NOTE: With -calcc- and -calcs- a blank tag entry (,,) means no calculation is done for the corresponding value of the conditional expression.

zero sets to zero a single variable or consecutive variables

```
zero    VAR
zero    STARTING VAR,NUM VARS
zero    (B)   (sets all defined variables to 0)
```

Note: In the 2-argument form, the number of bits zeroed is determined by the type designation of STARTING VAR.

set sets values of consecutive variables starting at the specified variable, or sets values of consecutive array elements starting at the specified element

```
set     STARTING VAR←EXPR1,EXPR2,EXPR3,...   (up to 95 values)
```

Note: All variables must be the same type as STARTING VAR.

compute evaluates a character string containing an expression and stores the result in the specified variable; the end of the string is determined by the specified number of characters, by a comma, by a semicolon, or by a zero byte (#00)

compute VAR FOR RESULT,STARTING VAR OF STRING,NUM 8-BIT BYTES IN STRING

Note: The string can contain up to 127 characters. Variables are not allowed, but these operators and system functions are permitted: + - ÷ (and /) × (and *) ** (and superscript) = ≠ < > ≤ ≥ parentheses and brackets π °
abs sqrt alog log exp ln sin cos tan cot sec csc
arcsin arccos arctan arccot arcsec arccsc

"zretinf" contains the number of characters evaluated, including terminating punctuation but not terminating zero byte

zreturn = -1 if the expression is evaluated successfully
= 0 if the expression contains operations when -specs noops- is in effect
= 1 if the expression contains an invalid character
= 2 if there are too many decimal points
= 3 if the expression is too complicated
= 4 if there is an unrecognized operator
= 5 if the expression has bad form
= 6 if there are unbalanced parentheses
= 7 if the expression contains undefined words
= 8 if a function contains an illegal argument

block copies a block of consecutive variables into another block of consecutive variables

block FROM STARTING VAR,TO STARTING VAR,NUM VARS

Note: The number of 8-bit bytes copied is determined by the type designation of the "from" variable.

Operations and symbols used in calculations

X + Y	addition
X - Y	subtraction
X × Y or X * Y	multiplication
X ÷ Y or X/Y	division
X $divr$ Y	integer division, result rounded to nearest integer
X $divt$ Y	integer division, result truncated to nearest integer
X**Y or X^Y	exponentiation (if exponent contains more than 1 character, superscript requires SHIFT SUPER before exponent and SHIFT SUB after exponent)
( ), [ ], { }	parentheses, brackets
¢	assignment of a value to a variable [e.g., var¢2.2×def+.45]

π = pi = 3.14159...

° = degree symbol [e.g., 53.4°]
number × 1° converts number to radians [e.g., sin(30×1°) or sin(30°)]
number ÷ 1° converts number to degrees [e.g., arcsin(.5)/1°]

16# or # prefix designates a hexadecimal constant [e.g., 16#4e or #4e]

2# prefix designates a binary constant [e.g., 2#0100 1110]

8# prefix designates an octal constant [e.g., 8#116]

" " used to place a character in the right-most 8 bits of an integer variable [e.g., var¢"s"]

Precedence of operations (in brief)

operations within parentheses
exponentiation
multiplication and division
addition and subtraction

Parentheses can be used to insure the desired order of operations.

Representation of numbers

Numbers are represented in twos-complement form; i.e., -X = comp(X)+1 .

The left-most bit of a signed integer is the sign bit (0 if integer ≥ 0, 1 if integer is < 0).
Range of values for 8-bit signed integers is -2^7 to $+(2^7 - 1)$, or -128 to +127 .
Range of values for 16-bit integers is -2^{15} to $+(2^{15} - 1)$, or -32768 to +32767 .
Range of values for 8-bit unsigned integers is 0 to $(2^8 - 1)$, or 0 to 255 .

Floating-point numbers contain 48 bits:
left-most bit is the sign bit (0 if number ≥ 0, 1 if number < 0);
next 15 bits contain the exponent;
right-most 32 bits contain the coefficient.
Values of floating-point numbers range from $\pm 2^{-16384}$ to $\pm 2^{+16383}$.
Floating-point numbers have a maximum of 10 significant digits.

System functions (argument can be an expression where appropriate)

abs(X)	absolute value of X	
frac(X)	fractional part of X	X is first rounded to the nearest integer if X is within about 10^{-9} of the integer
int(X)	integer part of X	
sqrt(X)	square root of X	

alog(X)	common antilogarithm of X (10^X)
log(X)	common logarithm of X (base 10 logarithm)
exp(X)	e^X (natural antilogarithm of X)
ln(X)	natural logarithm of X (base e logarithm)

With the following trigonometric functions, X is in radians. For X in degrees, the argument must be X°, e.g., sin(X°).

sin(X)	sine of X
cos(X)	cosine of X
tan(X)	tangent of X
cot(X)	cotangent of X
sec(X)	secant of X
csc(X)	cosecant of X

With the following inverse trigonometric functions, the result is in radians. For result in degrees, the function must be divided by 1°, e.g., arctan(X)/1° .

arcsin(X)	inverse sine, principal values $-\pi/2$ to $+\pi/2$
arccos(X)	inverse cosine, principal values 0 to $+\pi$
arctan(X)	inverse tangent, principal values $-\pi/2$ to $+\pi/2$
arccot(X)	inverse cotangent, principal values 0 to $+\pi$
arcsec(X)	inverse secant, principal values 0 to $+\pi$
arccsc(X)	inverse cosecant, principal values $-\pi/2$ to $+\pi/2$

zk(KEYNAME)	ASCII code for KEYNAME, e.g., zk(M), which has value #4d or 77_{10} ; zk(back), which has value #102 or 258_{10} ; KEYNAME must be specified; expression is not allowed; allowed keynames are given in the tables in the appendix
zlength(NAME)	number of elements in the array named NAME
zvloc(X)	absolute memory location in RAM of the variable X

Logical operations and functions (logical "true" is -1; logical "false" is 0)

X = Y	equal to	equality is "true" if $\lvert X-Y\rvert < (10^{-8} \times \lvert X\rvert)$ (approximately)
X ≠ Y	not equal to	
X ≤ Y	less than or equal to	
X ≥ Y	greater than or equal to	
X < Y	less than	
X > Y	greater than	
X and Y	logical "and"; result is "true" only if both X and Y are "true"	
X or Y	logical "or"; result is "true" if either X or Y or both are "true"	
not(X)	not(X) = 0 if X ≤ 0; not(X) = -1 if X > 0	

Operations which involve comparisons of floating-point numbers have a tolerance of 2^{-26} relative difference (approximately 1.5×10^{-8}).

Bit operations and functions (use with 16-bit integers) (a bit which is set = 1)

With shift operations (rsh, lsh, ars, cls), the shift must be between 0 and 16 (i.e., $0 \leq Y \leq 16$).

X rsh Y	shifts X to the right Y bit positions; bits shifted off the right end of X are dropped
X lsh Y	shifts X to the left Y bit positions; bits shifted off the left end of X are dropped
X ars Y	shifts X to the right Y bit positions; bits shifted off the right end of X are dropped; sign bit of X is copied into vacated bits on the left end of X
X cls Y	shifts X to the left, circularly, Y bit positions; bits shifted off the left end of X are copied into bits on the right end of X
X $mask$ Y	sets bits where bits are set in both X ano Y
X $unicn$ Y	sets bits where bits are set in either X or Y or both
X $diff$ Y	sets bits where bits are set in either X or Y but not both
comp(X)	ones complement of X (bit reversal)
bitcnt(X)	number of bits set in X

Random numbers

randu selects a random integer, sampled with replacement, and places it in the specified integer variable

randu VAR,MAXIMUM (selects an integer from 1 to MAXIMUM; $0 \leq MAXIMUM \leq (2^{14} - 1)$)

Note: If the integer selected is larger than the specified variable type can store, only the right-most eight bits are stored.

setperm creates a permutation list of integers of the specified length for sampling by the -randp- command

setperm LIST LENGTH,STARTING VAR OF LIST ($0 \leq LENGTH \leq (2^{14}-1)$;
first variable of the list contains the number of integers not yet selected from the list;
in succeeding variables each bit corresponds to an integer in the list and is 1 if the integer has <u>not</u> been selected, 0 if the integer has been selected;
integer variables are required:
8-bit signed variables if LENGTH < 128;
8-bit unsigned variables if LENGTH < 256;
16-bit variables if LENGTH ≥ 256);
number of variables required: 2 + int[(LENGTH - 1)/N],
where N is 8 or 16 for 8- or 16-bit variables respectively)

randp selects an integer, sampled without replacement, from the list set up by -setperm- or by an equivalent method, and places it in the specified integer variable; when the list is exhausted, the variable is set to 0

randp VAR FOR STORING VALUE,STARTING VAR OF LIST

Note: If the integer selected is larger than the specified variable type can store, only the right-most eight bits are stored.

remove removes the specified value from a permutation list

remove INTEGER TO REMOVE,STARTING VAR OF LIST

restore restores the specified value to a permutation list

restore INTEGER TO RESTORE,STARTING VAR OF LIST

Character manipulation

pack packs a character string starting in the specified integer variable; packs each character code into one 8-bit byte; if the byte count is not desired, the field is blank; string can contain embedded -show- and -showa- (and -showt- if the value is a floating-point variable)

```
pack    STARTING VAR FOR STORING STRING,VAR FOR STORING BYTE COUNT$
        STRING
pack    STARTING VAR FOR STORING STRING,$STRING
```

packc packs one of several character strings into a buffer of integer variables, depending on the rounded value of a conditional expression; packs each character code into one 8-bit byte; if byte count is not desired, the field is blank; string can contain embedded -show- and -showa- (and -showt- if the value is a floating-point variable)

```
packc   EXPR,STARTING VAR FOR STORING STRING,VAR FOR STORING
        BYTE COUNT$STRINGM$STRING0$STRING1$STRING2$$STRING4
```

search searches a buffer for the first occurrence of the specified character string (each character code occupies an 8-bit byte)

```
search  OBJ,OLEN,BUF,BLEN,STRT,POSN
```

OBJ = variable which contains the first character of the string to be found
OLEN = number of 8-bit bytes in the string to be found
BUF = starting variable of the buffer to be searched
BLEN = number of 8-bit bytes in the buffer to be searched
STRT = relative byte position in the buffer at which to start searching (1st position is 1; 2nd postion is 2; etc.)
POSN = variable for storing the relative byte position in the buffer where the object is found (0 if found in the first 8-bit byte, 1 if found in the second 8-bit byte, etc., -1 if not found)

searchf searches a buffer for the first occurrence of a character string in a specific field within an object

```
searchf OBJ,OLEN,BUF,ENTR,STRT,ELEN,BYTE,POSN

        OBJ  = variable which contains the first character of the
               string to be found
        OLEN = number of 8-bit bytes in the string to be found
        BUF  = starting variable of the buffer to be searched
        ENTR = number of entries in the buffer to be searched
        STRT = entry in the buffer at which to start searching
        ELEN = number of 8-bit bytes in each entry in the buffer
        BYTE = starting byte position within each entry for comparison
               with the object string (1st position is 1;
               2nd position is 2; etc.)
        POSN = variable for storing the relative position in the
               buffer of the entry where the object is found
               (0 if found in the first entry, 1 if found in the
               second entry, etc., -1 if not found)
```

Operations on lists

find searches each variable in a list of consecutive variables for the first occurrence of the specified object

```
find    OBJ,LIST,LEN,LOC

        OBJ  = variable containing the object of the search
        LIST = starting variable of the list (variables in the list
               must be the same type as the object)
        LEN  = number of variables in the list
        LOC  = variable for storing the relative location in the list
               where the object is found (0 if found in first variable,
               1 if found in second variable, etc., -1 if not found)
```

System variables for calculating

zretinf set by execution of -compute-; contains the number of characters evaluated, including terminating punctuation but not terminating zero byte

Additional notes on CALCULATING

Additional notes on CALCULATING

Additional notes on CALCULATING

F [redacted]

FILE OPERATIONS

NOTE: Attributes of files:
File names: up to 29 characters (letters and numerals) followed by a period and a two-letter extension (up to 32 characters total)

extensions:		
	.bi	lesson binary file
	.ch	charset
	.da	dataset (includes lesson files)
	.di	directory
	.mi	microtable
	.na	nameset
	.nf	notesfile

Record size: 128 8-bit bytes
Extra information: up to 32 8-bit bytes of extra information
Address: File Information Packet (FIP) number; up to 3 files can be addressed simultaneously by the FIP number (FIP can have value 1, 2, or 3)

In all commands in this section, FILE NAME can be a literal or the starting 8-bit variable of a buffer containing the file name. A variable name must be enclosed in parentheses: (STARTING VAR OF BUFFER) .

<u>All commands set "zreturn". Values of "zreturn" are given on page F10.</u>

Datasets

NOTE: A dataset is created as a "new" dataset or an "old" dataset. <u>New</u> datasets can have write and read codewords, but codewords are not required. <u>Old</u> datasets cannot have codewords.

setfile attaches the specified file to the specified FIP

```
setfile FIP;FILE NAME   (read and write access for a dataset without
                         codewords)
setfile FIP;FILE NAME,rw,(STARTING VAR FOR WRITE CODE) (read and write)
setfile FIP;FILE NAME,ro,(STARTING VAR FOR READ CODE)  (read only)
```

addfile creates a dataset and attaches it to the specified FIP

<u>old dataset</u>

```
addfile FIP;dataset;name,FNAME;numrecs,NRECS
```

<u>new dataset</u>

```
addfile FIP;datasetc;name,FNAME;numrecs,NRECS;writecode,(WCODE);
          readcode,(RCODE)
```

FNAME = file name, literal or (starting variable of a buffer)
NRECS = number of records in the file
WCODE = starting variable of buffer containing write codeword
RCODE = starting variable of buffer containing read codeword

delfile destroys the dataset attached to the specified FIP

delfile FIP

chgfile changes parameters of the dataset attached to the specified FIP

chgfile FIP;OPTION1;OPTION2;...

OPTIONS include:

name,FILE NAME or name,(STARTING VAR OF BUFFER)
info,(STARTING 8-BIT VAR OF BUFFER WITH NEW EXTRA INFO WITH FILE)
writecode,(STARTING VAR OF BUFFER WITH WRITE CODEWORD) (new dataset)
readcode,(STARTING VAR OF BUFFER WITH READ CODEWORD) (new dataset)

getfile stores parameters of the dataset attached to the specified FIP

getfile FIP;NAMEVAR,INFOVAR

NAMEVAR = starting 8-bit variable for storing file name
INFOVAR = starting 8-bit variable of buffer for storing file extra information

datain transfers data from records on the disk to the specified buffer

datain FIP;STARTING RECORD,STARTING VAR OF BUFFER,NUM RECORDS

dataout transfers data from the specified buffer to records on the disk

dataout FIP;STARTING RECORD,STARTING VAR OF BUFFER,NUM RECORDS

reserve sets "zreturn" in order to allow the user to reserve the dataset to prevent changes by more than one user at a time

reserve FIP

Note: If the dataset is reserved elsewhere, "zretinf" contains the port number where the dataset is reserved.

release sets "zreturn" to allow the dataset to be released

release FIP

Namesets

NOTE: A nameset is created as an alphabetized nameset or a nonalphabetized nameset. An alphabetized nameset is a set of named records arranged in alphabetical order. A nonalphabetized nameset is a set of named records arranged in an order specified by the user.

Parameters set by -addfile- when the nameset is created:

space authorization (spaceauth): number of records allocated (or authorized) on the disk for the entire nameset

maximum number of names (maxnames): maximum number of names allowed

length of each name (namelth): maximum number of characters in a name (range 1 to 32)

length of extra information with each name (infolth): maximum number of 8-bit bytes (range 0 to 32)

setfile attaches the specified file to the specified FIP

```
setfile FIP;FILE NAME   (read and write access for a nameset without
                         codewords)
setfile FIP;FILE NAME,rw,(STARTING VAR FOR WRITE CODE) (read and write)
setfile FIP;FILE NAME,ro,(STARTING VAR FOR READ CODE)  (read only)
```

addfile creates a nameset and attaches it to the specified FIP

<u>alphabetized nameset</u>

```
addfile FIP;nameset;name,FNAME;maxnames,NNAMES;spaceauth,NRECS;
        namelth,NAMLEN;infolth,INFLEN;writecode,(WCODE);readcode,(RCODE)
```

<u>nonalphabetized nameset</u>

```
addfile FIP;namesetn;name,FNAME;maxnames,NNAMES;spaceauth,NRECS;
        namelth,NAMLEN;infolth,INFLEN;writecode,(WCODE);readcode,(RCODE)
```

FNAME = file name, literal or (starting variable of a buffer)
NNAMES = number of names allowed in the nameset
NRECS = number of disk records allocated for the nameset
NAMLEN = number of characters in a name
INFLEN = number of bytes of extra information with a name
WCODE = starting variable of buffer containing write codeword
RCODE = starting variable of buffer containing read codeword

delfile destroys the nameset attached to the specified FIP; the nameset must be empty

```
delfile FIP
```

chgfile changes parameters of the nameset attached to the specified FIP

```
chgfile FIP;OPTION1;OPTION2;...
```

OPTIONS include:

```
name,NEW FILE NAME  or  name,(STARTING VAR OF BUFFER)
info,(STARTING 8-BIT VAR OF BUFFER WITH NEW EXTRA INFO WITH FILE)
writecode,(STARTING VAR OF BUFFER WITH NEW WRITE CODE)
readcode,(STARTING VAR OF BUFFER WITH NEW READ CODE)
```

getfile stores parameters of the nameset attached to the specified FIP

```
getfile FIP;NAMEVAR,INFOVAR

        NAMEVAR = starting 8-bit variable for storing file name
        INFOVAR = starting 8-bit variable of buffer for storing extra
                  information with file
```

names reads names (and extra information with each name) in the nameset attached to the specified FIP; stores data in the specified buffer

```
names   FIP;NAMEPOS,BUFVAR,BUFLEN

        NAMEPOS= numerical position of first name to store
        BUFVAR = starting 8-bit variable of buffer for storing names
                 and extra information with each name; format:
                    zfbpn(FIP) bytes:  first name;
                    zfbpi(FIP) bytes:  information with first name;
                    zfbpn(FIP) bytes:  second name;
                    zfbpi(FIP) bytes:  information with second name;
                    etc.
         BUFLEN = number of 8-bit bytes in the buffer
```

Note: "zretinf" is set to the number of names (with their associated extra information) stored in the buffer.

setname selects a name in the nameset attached to the specified FIP

```
setname FIP;NAME      (name is a literal)
setname FIP;(STARTING VAR FOR BUFFER CONTAINING NAME)
setname FIP;<first>  (selects the first name in the nameset)
setname FIP;<last>   (selects the last name in the nameset)
setname FIP;<next>   (selects the next name in the nameset)
setname FIP;<prev>   (selects the previous name in the nameset)
```

Note: "zretinf" is set to the number of partial matches to the specified name.

addname adds a new name and its records to the nameset attached to the specified FIP

<u>alphabetized nameset</u>
addname FIP;NAME,NRECS,INFOVAR

<u>nonalphabetized nameset</u>
addname FIP;POSN,NAME,NRECS,INFOVAR

POSN = position of new name; value from 1 to zfmaxn(FIP)
NAME = name, literal or (starting variable of a buffer)
NRECS = number of records in the named set of records
INFOVAR = starting 8-bit variable of buffer with extra information with name

delname destroys the selected name and its records

delname FIP

chgname changes parameters of the selected name

chgname FIP;OPTION1;OPTION2;...

OPTIONS include:

name,NEW NAME <u>or</u> name,(STARTING VAR OF BUFFER)
info,(STARTING 8-BIT VAR OF BUFFER WITH NEW EXTRA INFO WITH NAME)
position,NEW POSITION (value from 1 to zfmaxn(FIP); valid only with nonalphabetized namesets)

getname stores parameters of the selected name

getname FIP;NAMEVAR,INFOVAR

NAMEVAR = starting 8-bit variable for storing name
INFOVAR = starting 8-bit variable of buffer for storing extra information with name

addrecs adds records to the selected name starting at the specified position

addrecs FIP;STARTING RECORD POSITION,NUM RECORDS TO ADD

delrecs deletes records from the selected name starting at the specified position

delrecs FIP;STARTING RECORD POSITION,NUM RECORDS TO DELETE

datain transfers data from disk records for the selected name to a buffer

datain FIP;STARTING RECORD,STARTING VAR OF BUFFER,NUM RECORDS

dataout transfers data from a buffer to disk records for the selected name

dataout FIP;STARTING RECORD,STARTING VAR OF BUFFER,NUM RECORDS

reserve sets "zreturn" in order to allow the user to reserve the nameset to prevent changes by more than one user at a time

reserve FIP

Note: If the dataset is reserved elsewhere, "zretinf" contains the port number where the dataset is reserved.

release sets "zreturn" to allow the nameset to be released

release FIP

Directories

NOTE: A directory is a type of file that contains file names.

setfile attaches the specified file to the specified FIP

```
setfile FIP;FILE NAME   (read and write access for a directory without
                         codewords)
setfile FIP;FILE NAME,rw,(STARTING VAR FOR WRITE CODE) (read and write)
setfile FIP;FILE NAME,ro,(STARTING VAR FOR READ CODE)  (read only)
```

addfile creates a directory and attaches it to the specified FIP

```
addfile FIP;directory;name,DNAME;maxnames,NFILES;spaceauth,NRECS;
        namelth,NAMLEN;infolth,INFLEN;writecode,(WCODE);readcode,(RCODE)

        DNAME = directory name, literal or (starting variable of buffer)
        NFILES= number of files allowed in the directory
        NRECS = number of disk records allocated for all files in the
                directory
        NAMLEN= number of characters in file names in the directory
                (value from 4 to 32, including 3-character extension)
        INFLEN= number of bytes of extra information with each file
                in the directory (value from 0 to 32)
        WCODE = starting variable of buffer containing write codeword
        RCODE = starting variable of buffer containing read codeword
```

delfile destroys the directory attached to the specified FIP; the directory must be empty

```
delfile FIP
```

chgfile changes parameters of the directory attached to the specified FIP

```
chgfile FIP;OPTION1;OPTION2;...
```

OPTIONS include:

name,NEW DIRECTORY NAME <u>or</u> name,(STARTING VAR OF BUFFER)
infc,(STARTING 8-BIT VAR OF BUFFER WITH NEW EXTRA INFO WITH DIRECTORY)
writecode,(STARTING VAR OF BUFFER WITH NEW WRITE CODE)
readcode,(STARTING VAR OF BUFFER WITH NEW READ CODE)

getfile stores parameters of the directory attached to the specified FIP

getfile FIP;NAMEVAR,INFOVAR

NAMEVAR = starting 8-bit variable for storing directory name
INFOVAR = starting 8-bit variable of buffer for storing extra information with directory

names reads file names (and extra information with each name) in the directory attached to the specified FIP; stores data in the specified buffer

names FIP;NAMEPOS,BUFVAR,BUFLEN

NAMEPOS= numerical position of first file name to store
BUFVAR = starting 8-bit variable of buffer for storing file names and extra information with each file name;
zfbpn(FIP) bytes: first file name;
zfbpi(FIP) bytes: information with first name;
zfbpn(FIP) bytes: second file name;
zfbpi(FIP) bytes: information with second name;
etc.
BUFLEN = number of 8-bit bytes in the buffer

Note: "zretinf" is set to the number of file names (with their associated extra information) stored in the buffer.

setdir specifies a directory for use with -setfile- and -addfile- commands

setdir DIRECTORY NAME
setdir (STARTING VAR OF BUFFER CONTAINING NAME)
setdir <*> (system default directory)

Note: Files in a directory can be addressed without a preceding -setdir- command; a "path" is specified.

For example:
<*>/root.di/analysis.di/class.da
<*>/main.di/characters.ch

reserve sets "zreturn" in order to allow the user to reserve the directory to prevent changes by more than one user at a time

reserve FIP

Note: If the dataset is reserved elsewhere, "zretinf" contains the port number where the dataset is reserved.

release sets "zreturn" to allow the directory to be released

release FIP

System variables and functions for file operations

These functions are set for the file attached to the specified FIP.
FIP can have value of 1, 2, or 3.

zfauth(FIP) number of records allocated (or authorized) on the disk for the attached nameset or directory

zfbpi(FIP) number of 8-bit bytes of extra information for each name in the attached nameset or directory

zfbpn(FIP) number of characters (8-bit bytes) in each name in the attached nameset or directory

zfbpr(FIP) number of 8-bit bytes in each record in the attached file (= 128)

zfmaxn(FIP) maximum number of names allowed in the attached nameset or directory

zfnams(FIP) number of names in the attached nameset or directory

zfrecs(FIP) number of records in the attached file

zftype(FIP) value designating type of the attached file
= 0 for a directory
= 1 for a dataset without codewords
= 2 for an alphabetized nameset
= 4 for a nonalphabetized nameset
= 5 for a dataset with codewords

zretinf set by execution of -names-, -reserve-, -setname-
-names-: contains the number of names stored in the buffer
-reserve-: contains the port number where the attached file is reserved
-setname-: contains the number of partial matches to the specified name ("zreturn" has value 0)

("zreturn" values are given on the next page)

zreturn set according to results of an operation

= -1 if the operation is successful
= 0 if the specified name matches more than one name in the attached nameset (selects the first partial match)
= 1 if a name has illegal form
= 2 if the FIP number is not 1, 2, or 3
= 3 if the appropriate file type is not attached or if no name has been selected in a nameset
= 4 if the file, directory, or name does not exist
= 5 if the attached file is not the correct type for the requested operation
= 6 if the codeword argument is omitted or does not match the file codeword
= 7 if the file or name already exists (duplicate name)
= 8 if sufficient disk space is not available for the operation
= 9 if a parameter has illegal value (e.g., out of range)
= 10 if space is not available for more names (nameset or directory is full)
= 11 if write access has not been granted
= 12 if the file is in use elsewhere on the system
= 13 if the nameset or directory is not empty
= 14 if record numbers extend out of range
= 15 if reservation request and reservation status conflict: the file is reserved elsewhere (for -reserve-) or the file is not reserved (for -release-)
= 16 if name positions extend beyond the end of the nameset or directory
= 17 (system error) system crashed
= 18 if the directory has been altered
= 19 if the required directory is too large
= 50 (system error) illegal request
= 51 (system error) lesson buffer space not available
= 52 (system error) physical disk space not available
= 53 (system error) disk read/write error

Additional notes on FILE OPERATIONS

Additional notes on FILE OPERATIONS

JUDGING

J

Preparation for responding

darrow (non-executable) establishes a buffer (starting variable and number of 8-bit bytes) for all subsequent -arrow- commands; if -darrow- is omitted, the buffer must be specified with the -arrow- command

darrow STARTING VAR,NUM 8-BIT BYTES (maximum of 150 bytes)

arrow places an arrow on the screen at the specified location and collects keyset input in the specified buffer; indented commands which follow -arrow- are executed before processing stops to wait for input; non-indented commands which follow these indented commands are executed each time a judging key is pressed to initiate judging

arrow LOCATION;STARTING VAR,NUM 8-BIT BYTES (maximum of 150 bytes; LOCATION can be COARSE <u>or</u> FINEX,FINEY)
arrow LOCATION (buffer established by preceding -darrow-)

endarrow (no tag) must terminate response processing; if the response is matched, indented commands following the matched response and indented commands following -ifmatch- are executed; if the "wrong" response is matched <u>or</u> if the response is not matched, judgment is "no" and processing stops until another response is entered; if judgment is "ok", response processing is complete and commands following -endarrow- are executed

long modifies the maximum number of character codes allowed at an arrow set by -arrow- or by -darrow-; reset at each -arrow-

long NUM 8-BIT BYTES (maximum of 150 bytes)

force alters the input of a response as specified; cleared at each main unit

force caps (converts lower-case letters [a through z] to upper-case letters [A through Z]; also affects a letter entered at a -pause-)
force firsterase (erases an incorrect response and contingent message when the user presses any key)
force font (displays characters in alternate font)
force long (initiates judging when the number of characters entered reaches the limit set by -long-)
force left (writes response from right to left in alternate font)
force micro (substitutes microtable definition for each keypress)
force full (displays characters which are 16 dots wide by 24 dots high)
force hira (displays hiragana characters)
force kata (displays katakana characters)
force (B) or force clear (clears the current setting of -force- in this unit)
force clear,font (can combine tags)

Note: Tags "full", "hira", and "kata" are available on stations offering these features.

eraseu the specified unit is executed at all subsequent arrows in the unit containing -eraseu- when the user erases all or part of a response after receiving judgment; does not alter default judge-process erasing; remains in effect until reset or until a new unit is executed

eraseu UNIT NAME
eraseu q (clears -eraseu- setting for remainder of the unit)
eraseu EXPR,NAMEM,NAME0,q,NAME2,x (example of conditional form; argument q clears setting; argument x leaves setting unchanged)

jkey specifies keys (in addition to NEXT) which initiate judging; cleared at each -arrow-; a judging key which is not a function key appears as the last key in the response buffer unless -inhibit jkeys- is in effect; names listed in the -keylist- command are permitted, including system-defined keylist names

jkey KEY1,KEY2,KEY3 (e.g., jkey back,=,a)

Note: -jkey touch- automatically sets -enable touch-.

copy activates COPY key and specifies a buffer containing characters to be written on the screen one word at a time when COPY is pressed; loads the string into the response buffer exactly as it appears on the screen; cleared at each -arrow-

copy STARTING VAR OF COPY BUFFER,NUM 8-BIT BYTES

Modification of the response

putd replaces a character string in the response buffer with another character string; the first character in the tag is interpreted as the delimiter between strings

```
putd    /STRING1/STRING2/   (delimiter is /)
putd    ,STRING1,STRING2,   (delimiter is ,)
```

```
Note:  zreturn = -1  if -putd- is executed successfully
               =  0  if the replacement string would cause the
                     response to be longer than the storage buffer
```

Modification of the judging procedure

specs modifies standard judging procedures for all subsequent answer processing at that arrow; settings are cumulative at an arrow; cleared at each -arrow- command

specs	nomark	(prevents default answer markup)
specs	nookno	(prevents appearance of "ok" and "no")
specs	noops	(prevents use of mathematical operators in a numerical response)
specs	nospell	(turns off default spelling checks; no spelling markup is done; "zspell" is not set)
specs	okcap	(allows capitalized word in the response to match a non-capitalized word in the tag of a response-matching command)
specs	okextra	(allows "extra" words in the response, i.e., words not in the tag of the response-matching command)
specs	okspell	(allows any reasonable spelling of words in the response)
specs	punc	(allows only punctuation specified in the response-matching command; without -specs punc-, specified punctuation must be present, but additional punctuation may also be present)
specs	(B)	(clears previous settings at this arrow)
specs	nookno,okcap,okspell	(can combine tags)

Matching the response

NOTE: With the following commands (-keyword-, -answer-, -wrong-, -answerc-, -wrongc-, -exact-, -exactw-, -ansv-, -wrongv-) if the response matches the tag or the required argument, subsequent indented commands are executed up to the next non-indented command.

With -answer-, -wrong-, -answerc-, -wrongc-:
The separator between words is a space.
Punctuation symbols are , . ? ! ; : /
Up to 40 required words are permitted in the tag. Up to 50 words can be entered by the student.

Response markup symbols:

===	word is misspelled
↑	word is capitalized incorrectly
⇕	word is out of order (too far right)
^	word is missing
xxx	word is an extra word

keyword checks the response for words listed in the tag; if a word is matched, the variable is set to the relative position in the tag of the matched word and judgment is "ok" ("zjudged" set to -1); if no word is matched, the variable is set to -1, judgment is not made, and judging continues; a maximum of 50 words can be specified in the tag

```
keyword VAR⇕WORD0⇕[WORD1 SYNONYM1]⇕WORD2⇕⇕WORD4
```

answer compares the response with the -answer- tag; checks for spelling, capitalization, extra words, and punctuation unless altered by -specs-; punctuation marks are treated as words; sets "zjudged" to -1 if the response matches the tag

```
answer  <EXTRA WORDS> [WORD1 SYNONYM1] WORD2 WORD3
            (blank tag matches a response in which nothing is entered
            or which contains only spaces and punctuation;
            -allow  blanks- must be in effect)
answer  <a,STARTING VAR,NUM 8-BIT BYTES>
```

wrong similar to -answer- but for an incorrect response; sets "zjudged" to 0 if the response matches the tag

```
wrong   <EXTRA WORDS> [WORD1 SYNONYM1] WORD2 WORD3
wrong   <a,STARTING VAR,NUM 8-BIT BYTES>
```

answerc conditional form of -answer-; performs checks available with -answer-; sets "zjudged" to -1 if the response matches the required argument

```
answerc EXPR‡RESPONSEM‡RESPONSE0‡‡RESPONSE2
```

wrongc similar to -answerc- but for an incorrect response; sets "zjudged" to 0 if the response matches the required argument

```
wrongc  EXPR‡RESPONSEM‡RESPONSE0‡RESPONSE1‡‡RESPONSE3
```

exact compares the response with the tag for an exact character by character match; sets "zjudged" to -1 if the response matches the tag

```
exact   STRING   (blank tag matches a response in which nothing is
                 entered; -allow  blanks- must be in effect)
exact   ‹a,STARTING VAR,NUM 8-BIT BYTES›
```

exactw similar to -exact- but for an incorrect response; sets "zjudged" to 0 if the response matches the tag

```
exactw  STRING
exactw  ‹a,STARTING VAR,NUM 8-BIT BYTES›
```

ansv checks a numerical response against the first argument in the tag, with tolerance set by the optional second argument; sets "zjudged" to -1 if the response matches the tag within the tolerance; tolerance can be stated as absolute deviation or percent deviation; if tolerance is omitted, the response value must match the tag value

```
ansv    EXPR,TOLERANCE
```

wrongv similar to -ansv- but for an incorrect numerical response; sets "zjudged" to 0 if the response matches the tag within the tolerance

```
wrongv  EXPR,TOLERANCE
```

or (no tag) placed on the line between response-matching commands to provide alternative responses; if the tag of any command linked by -or- is matched, indented commands following the last linked response-matching command are executed

ok judges a response "ok" and sets "zjudged" to -1 if the rounded value of the tag is negative; if the judgment is "ok", indented commands following -ok- are executed

ok EXPR (blank tag is equivalent to negative value)

no judges a response "no" and sets "zjudged" to +1 if the rounded value of the tag is negative; if the judgment is "no", indented commands following -no- are executed

no EXPR (blank tag is equivalent to negative value)

ifmatch (no tag) indented commands following -ifmatch- are executed whenever a response is matched, independent of judgment ("zjudged" equals -1, 0, or +1); only one -ifmatch- can occur for each -arrow-; -ifmatch- must be the last non-indented command before -endarrow-

Information on specific words in the response

getmark used after judging a response to give markup information on individual words in the response

getmark POSN,MARKUP

POSN = relative position of the word in the response (first word is 1, second word, 2, etc.)
MARKUP = variable containing markup information (must be 16-bit integer variable)

= -2 if the response is perfect or if no markup is done with the response-matching command used
= -1 if the position of the word is out of bounds (i.e., if POSN > "zwcount")
= 0 if there are no errors in the word
> 0 bits in MARKUP are set according to the error(s), starting at the right-most bit (subscript "2" indicates binary notation):
(1_2) a word preceding this word is missing
(10_2) the word is out of order (too far right)
(100_2) the word is capitalized incorrectly
($1\ 000_2$) the word is spelled incorrectly
($10\ 000_2$) [bit not set]
($100\ 000_2$) the word is an extra word
($1\ 000\ 000_2$) this word is the last word, and a word which should follow is missing

getloc gives the screen position of the beginning (and end, if requested) of the specified word in the response

getloc POSN,XBEG,YBEG,*XEND,YEND*

POSN = relative position of the word in the response (first word is 1, second word, 2, etc.)
XBEG = variable for storing the finex screen position of the beginning of the word (= -1 if POSN > "zwcount")
YBEG = variable for storing the finey screen position of the beginning of the word
XEND = variable for storing the finex screen position of the end of the word (optional)
YEND = variable for storing the finey screen position of the end of the word (optional)

Reference to other units during judging

iarrow specifies the unit to be executed immediately after each subsequent -arrow- in a main unit; equivalent to indented -do- command after the -arrow- command; cleared at each main unit; later occurrence in the unit overrides an earlier setting in the unit

```
iarrow  UNIT NAME
iarrow  q   (clears previous setting in the unit)
iarrow  EXPR,UNITM,UNIT0,x,q,UNIT3  (example of conditional form;
           argument q clears setting; argument x leaves setting
           unchanged)
```

ijudge specifies the unit to be executed each time the user presses a judging key; equivalent to non-indented -do- command after -arrow- following indented commands but preceding response-matching commands; cleared at each main unit; later occurrence in the unit overrides an earlier setting in the unit

```
ijudge  UNIT NAME
ijudge  q   (clears previous setting in the unit)
ijudge  EXPR,UNITM,UNIT0,q,UNIT2,x  (example of conditional form;
           argument q clears setting; argument x leaves setting
           unchanged)
```

Alteration of judgment

judge alters the judgment rendered by judging commands

judge	ok	(sets judgment to "ok"; sets "zjudged" to -1; executes subsequent commands up to the next judging or non-indented command before branching to -ifmatch- [or -endarrow-])
judge	no	(sets judgment to "no" [unanticipated]; sets "zjudged" to +1; executes subsequent commands up to the next judging or non-indented command before branching to -ifmatch- [or -endarrow-]); returns to the arrow for additional input)
judge	wrong	(sets judgment to "no" [anticipated]; sets "zjudged" to 0; executes subsequent commands up to the next judging or non-indented command before branching to -ifmatch- [or -endarrow-]); returns to the arrow for additional input)
judge	okquit	(sets judgment to "ok"; sets "zjudged" to -1; branches to -ifmatch- [or -endarrow-])
judge	noquit	(sets judgment to "no"; sets "zjudged" to +1; branches to -ifmatch- [or -endarrow-]; returns to the arrow for additional input)
judge	quit	(does not alter judgment or "zjudged"; branches to -ifmatch- [or -endarrow-]; does not return to the arrow even if judgment is not "ok" and allows the student to leave the arrow)
judge	exdent	(sets "zjudged" to 2; branches to next non-indented command and continues looking for a match)
judge	exit	(sets "zjudged" to 2; returns to the arrow to wait for additional input)
judge	ignore	(sets "zjudged" to 2; stops processing, erases response, returns to the arrow for additional input)
judge	unjudge	(sets "zjudged" to 2; continues processing commands at the same level of indentation)
judge	rejudge	(sets "zjudged" to 2; restores the original response [unmodified by -putd-, etc.], initializes "zanscnt", "zwcount", and closest match pointer; performs -judge exdent- and continues looking for a match to the unmodified response)
judge	x	(leaves judgment unchanged; used in conditional form)
judge	EXPR,no,ok,x,wrong	(example of conditional form)

Alteration of feedback

okword changes "ok" message to the character string in the specified buffer (if tag is blank, message does not appear)

okword STARTING VAR OF "OK" MESSAGE,NUM 8-BIT BYTES

noword changes "no" message to the character string in the specified buffer (if tag is blank, message does not appear)

noword STARTING VAR OF "NO" MESSAGE, NUM 8-BIT BYTES

System variables for judging

zanscnt number of response-matching commands encountered at an arrow before the response is matched; = -1 if no tag is matched

zcaps = -1 if there are no capitalization errors, = 0 otherwise

zentire = -1 if all required words are present in the response, = 0 otherwise

zextra = -1 if there are no extra words in the response, = 0 otherwise

zjcount number of character codes in the response

zjudged = -1 for any "ok" judgment
= 0 for any "wrong" judgment (anticipated "no")
= 1 for any "no" judgment (unanticipated "no")
= 2 for a response which is not matched; also set by -judge exdent-, -judge exit-, -judge ignore-, -judge unjudge-, -judge rejudge-

zntries number of attempts at the current arrow

zopcnt number of arithmetic operations in a numerical response (set with -ansv-, -wrong v-, -compute-)

zorder = -1 if the word order is correct, = 0 otherwise

zspell = -1 if spelling is correct, = 0 otherwise

zwcount number of words in the response (maximum of 50); set by -answer-, -wrong-, -answerc-, -wrongc-

Additicnal notes on JUDGING

Additional notes on JUDGING

PRESENTING

P [redacted]

P

Screen size

coarse states the size (in dots) of a character on the screen and determines the meaning of coarse-grid coordinates

coarse NUMBER OF DOTS WIDE,NUMBER OF DOTS HIGH

fine states parameters of the screen for which the lesson was written; an optional rectangular region can be specified by giving coordinates of two opposite corners closest to and furthest from the screen origin; if a region is not specified, it is identical to the entire screen

fine XDOTS,YDOTS,XINC,YINC,*XMIN,YMIN,XMAX,YMAX*

XDOTS = horizontal size of the screen in dots
YDOTS = vertical size of the screen in dots
XINC = direction of horizontal increment:
right if x increases from left to right;
left if x increases from right to left
YINC = direction of vertical increment:
up if y increases from bottom to top;
down if y increases from top to bottom
XMIN = x coordinate of region corner closest to screen origin
YMIN = y coordinate of this corner
XMAX = x coordinate of region corner furthest from origin
YMAX = y coordinate of this corner

Note: zreturn = -1 if the region fits on the screen on which the lesson is executed
= 0 if the region does not fit

The region is centered on the screen on which the lesson is executed.

Coarse-grid coordinates always increase from top to bottom and in the direction of XINC.

Screen origin is at upper-left corner for XINC right, YINC down
lower-left corner for XINC right, YINC up
lower-right corner for XINC left, YINC up
upper-right corner for XINC left, YINC down

window establishes a rectangular window on the screen outside of which no display is plotted; remains in effect until reset; LOCATION is the screen location and can be COARSE <u>or</u> FINEX, FINEY; "zxmin", "zymin" and "zxmax", "zymax" are fine-grid coordinates of opposite corners of the window (closest to and furthest from the screen origin); if no window is declared, these coordinates are identical to the corners of the region established by a previous -fine- command

window CORNER LOCATION;OPPOSITE CORNER LOCATION
window ;CORNER LOCATION (opposite corner at "zwherex", "zwherey")
window (B) (resets to the region established by a previous -fine-)

Basic display

at specifies starting position of display on the screen; sets margin for display of text

```
at      COARSE
at      FINEX,FINEY
```

atnm like -at- but does not reset the margin

```
atnm    COARSE
atnm    FINEX,FINEY
```

write displays text, including embedded information

```
write   MESSAGE, INCLUDING EMBEDDED INFORMATION
```

writec displays one of several messages, depending on the value of the conditional expression; the conditional expression must conform to restrictions on calculations

```
writec  EXPR$MESSAGEM$MESSAGE0$MESSAGE1$$MESSAGE3
```

NOTE: The following embed features are available. See descriptions of the individual commands for definitions of the arguments.

```
{show,EXPR}                      or  {s,EXPR}
{showt,EXPR,LEFT,RIGHT}          or  {t,EXPR,LEFT,RIGHT}
{showb,EXPR,NUM BITS}            or  {b,EXPR,NUM BITS}
{showo,EXPR,NUM PLACES}          or  {o,EXPR,NUM PLACES}
{showh,EXPR,NUM PLACES}          or  {h,EXPR,NUM PLACES}
{showa,STARTING VAR,BYTE COUNT}  or  {a,STARTING VAR,BYTE COUNT}
{at,COARSE}; {at,FINEX,FINEY}
{atnm,COARSE}; {atnm,FINEX,FINEY}
```

show displays a value in decimal notation; displays up to 3 digits to the right of the decimal point and up to 10 digits total

```
show    EXPR
```

showt displays a value in decimal notation in the specified format

```
showt   EXPR,PLACES LEFT OF DECIMAL POINT,PLACES RIGHT OF DECIMAL POINT
            (format, if omitted, is 4,3; if third argument is omitted,
            no places are shown to the right of the decimal point)
```

showb displays an integer value in binary notation; displays the specified number of bits, counting from the right end of the value

showb EXPR,NUM BITS

showo displays an integer value in octal notation; displays the specified number of places, counting from the right end of the value

showo EXPR,NUM PLACES

showh displays an integer value in hexadecimal notation; displays the specified number of places, counting from the right end of the value

showh EXPR,NUM PLACES

showa displays characters in the specified integer variable(s), reading from the left end of the buffer; each character code is in an 8-bit byte

showa STARTING VAR,NUM 8-BIT BYTES

text displays contents of an alphanumeric buffer line by line; the end of a line must be indicated by an 8-bit byte equal to 0; not affected by -size- or -rotate-

text STARTING VAR,NUM 8-BIT BYTES TO DISPLAY

textn similar to -text- except lines of text are numbered to the left of each line; not affected by -size- or -rotate-

textn BUF,LEN,DISP,FNUM,LNUM,MAX

BUF = starting variable of the buffer which contains text
LEN = total number of 8-bit bytes to display
DISP = variable for storing the number of 8-bit bytes displayed plus 1 (not affected by the value of MAX)
FNUM = line number of the first line displayed (if equal to 0, no text is displayed)
LNUM = line number of the last line displayed (maximum is 31)
MAX = maximum number of 8-bit bytes to display per line (can be less than the number of bytes indicated by the placement of the end-of-line marker, an 8-bit byte equal to 0)

Note: zreturn = -1 if FNUM and LNUM are in the range 0 to 31
= 0 otherwise

NOTE: With -text- and -textn-, "zretinf" contains the number of lines displayed on the screen.

erase erases the screen, selectively or entirely

```
erase   (B)   (causes full-screen erase)
erase   NUM CHARACTERS TO ERASE
erase   NUM CHARACTERS PER LINE,NUM LINES
```

mode specifies display mode (see system variable "zmode")

```
mode    write     (normal writing state; writes selected dots)
mode    erase     (erases selected dots)
mode    rewrite   (erases and rewrites in one step)
mode    inverse   (displays dark characters on light background)
mode    EXPR,erase,write,x,inverse   (example of conditional form;
           argument x leaves writing mode unchanged)
```

Note: The mode is reset to "write" after any full-screen erase, in particular at a main unit. However, the mode is unchanged if the previous unit contained -inhibit erase-.

size specifies bold-face writing <u>or</u> sets size for relocatable commands (-rdraw-, -rcircle-, etc.)

```
size    SIZE   (does not affect writing)
size    SIZE IN X DIRECTION,SIZE IN Y DIRECTION
size    bold   (specifies bold-face writing)
size    0  or  size    (B)    (restores standard writing)
```

rotate sets angle for relocatable commands (-rdraw-, -rcircle-, etc.)

```
rotate  ANGLE IN DEGREES     (does not affect writing)
rotate  0  or  rotate  (B)   (restores normal display direction)
```

NOTE: Default settings in each main unit are: -inhibit blanks- and -allow arrow,display,erase,keys,jkeys- . These settings are in effect unless altered by -inhibit- and -allow-.

inhibit disables certain default actions in a unit; settings are cleared at each main unit and default settings are restored; effect within a unit is cumulative: i.e., later occurrence of -inhibit- is added to the effect of an earlier occurrence

inhibit arrow	(prevents plotting of the response arrow)
inhibit blanks	(prevents judging if a judging key is pressed before a response is entered; default setting)
inhibit display	(prevents plotting of a display but updates screen position "zwherex", "zwherey" as if plotting had occurred)
inhibit erase	(prevents full-screen erase when proceeding to a new main unit; retains status of "zwherex", "zwherey", "zmode", and -enable-)
inhibit jkeys	(prevents a non-function "jkey" from being stored as the last key in the response buffer)
inhibit keys	(prevents keyset input from breaking through -pause-)
inhibit (B)	(re-establishes the default settings in this main unit; equivalent to: -inhibit blanks- and -allow arrow,display,erase,jkeys,keys-

allow permits actions which have been inhibited in the unit by -inhibit-; effect within a unit is cumulative: i.e., later occurrence of -allow- is added to the effect of an earlier occurrence

allow arrow	(allows the response arrow to be plotted)
allow blanks	(allows null input at a response arrow; default is -inhibit blanks-)
allow display	(allows normal plotting of display)
allow erase	(allows a full-screen erase at a new main unit)
allow jkeys	(allows a non-function "jkey" to be stored as the last key in the response buffer)
allow keys	(allows keyset input to break through -pause-)
allow (B)	(establishes settings opposite to default settings; equivalent to: -allow blanks- and -inhibit arrow,display,erase,jkeys,keys-

Graphics

NOTE: With -dot-, -draw-, -box-, -fill-, -vector-, LOCATION is the screen location and can be COARSE or FINEX,FINEY. Coarse-grid and fine-grid coordinates can be mixed in tags with more than one argument.

dot — draws a dot at the specified screen location

```
dot     LOCATION
```

draw — draws a dot, line, or line-drawn figure; after execution, "zwherex" and "zwherey" are set to the last point plotted

```
draw    LOCATION   (equivalent to -dot-)
draw    LOCATION1;LOCATION2   (draws a line)
draw    LOCATION1;LOCATION2;LOCATION3   (draws connected lines)
draw    ;LOCATION   (draws a line from the current screen location to
           the specified location)
draw    LOCATION1;LOCATION2;skip;LOCATION3;LOCATION4
           ("skip" moves to a new position without plotting)
```

box — draws a rectangle with the specified corner locations and thickness; after execution, "zwherex", "zwherey" are set to the corner of the box (with thickness included) closest to the screen origin

```
box     CORNER LOCATION;OPPOSITE CORNER LOCATION;DOTS THICK
box     ;CORNER LOCATION;DOTS THICK  (opposite corner at current
                                      "zwherex", "zwherey")
box     (B)   (draws a rectangle with corners specified by previous
              -window- command or -fine- command; equivalent to
              -box  zxmin,zymin;zxmax,zymax-)
```

Note: Thickness, if omitted, 0, 1, or -1, is 1 dot. Negative thickness extends inward; positive thickness extends outward.

fill — fills (-mode write- or -mode rewrite-) or erases (-mode erase- or -mode inverse-) a rectangular area on the screen; does not affect the setting of "zwherex", "zwherey"

```
fill    CORNER LOCATION;OPPOSITE CORNER LOCATION
fill    ;CORNER LOCATION   (opposite corner at "zwherex", "zwherey")
fill    (B)   (fills in a rectangle with corners specified by previous
              -window- command or -fine- command; equivalent to
              -fill zxmin,zymin;zxmax,zymax-)
```

vector draws a vector symbol with specified tail and head locations and head size

vector TAIL LOCATION;HEAD LOCATION;*SIZE*
vector ;HEAD LOCATION;*SIZE* (tail at "zwherex", "zwherey")

Note: SIZE, if omitted, is 10 or 11 dots for moderate-length vectors. Negative size indicates open arrowhead. |size|≥1 is absolute (in screen dots); |size|<1 is relative to the length of the vector.

circle draws a circle with the specified parameters; the center is at the current "zwherex", "zwherey"; after execution, "zwherex", "zwherey" are set to the center for a one-argument tag and to the end of the last line drawn for the three-argument tag

circle RADIUS IN DOTS,*START ANGLE,END ANGLE*
(second and third arguments are optional: if omitted, START ANGLE is 0° and END ANGLE is 360°; degree symbol is omitted; angles are measured in degrees from START ANGLE, from positive x direction toward positive y direction; counter-clockwise if screen origin is at lower-left corner or upper-right corner of screen
clockwise if screen origin is at upper-left corner or lower-right corner of screen

circleb same ootions as -circle- but draws a broken circle

circleb RADIUS IN DOTS,*START ANGLE,END ANGLE*

Relocatable graphics

rorigin establishes a "relocatable" origin for use with -rat-, -ratnm-, -rdot-, -rdraw-, and -rcircle-; remains in effect across main unit boundaries until reset; initially set to -rorigin 0,0- upon entry to a lesson

rorigin FINEX,FINEY
rorigin (B) (sets relocatable origin to "zwherex", "zwherey")

NOTE: Subsequent relocatable commands are affected by preceding -rorigin-, -size-, -rotate-, and XINC and YINC in the -fine- command.

rat similar to -at- but relative to the -rorigin- location; affected by -size-, -rotate-, and -fine-

rat X-LOCATION,Y-LOCATION
rat (B) (equivalent to -rat 0,0-, i.e., the current -rorigin- location)

ratnm similar to -rat- but does not reset the left margin (see -atnm-)

ratnm X-LOCATION,Y-LOCATION

rdot draws a dot at the specified position relative to the -rorigin- location; affected by preceding -size-, -rotate-, and -fine-

rdot X-LOCATION,Y-LOCATION

rdraw similar to -draw- but figure is relative to the -rorigin- location; affected by preceding -size-, -rotate- and -fine-; "zwherex" and "zwherey" are set to the last point plotted

for example:

rdraw X1,Y1;X2,Y2 (draws a line relative to -rorigin-)

rcircle same options as -circle-; size and orientation are affected by -size- and -rotate-; direction of plotting is affected by -fine-; gives an ellipse if preceded by two-argument -size- with unequal arguments; the specified radius is the radius before being affected by -size- (see -circle-)

rcircle RADIUS IN DOTS,*START ANGLE,END ANGLE*

Drawing graphs

gorigin specifies location of the origin of the graph; all other display with graphing commands is relative to this origin; remains in effect across main unit boundaries until reset; initially set to -gorigin 0,0- upon entry to a lesson

gorigin FINEX,FINEY
gorigin (B) (sets graph origin to "zwherex", "zwherey")

axes specifies lengths of the axes and draws the axes; x and y values are in dots relative to the -gorigin- location

axes NEGATIVE X,NEGATIVE Y,POSITIVE X,POSITIVE Y
axes POSITIVE X,POSITIVE Y

Note: To draw one-quadrant axes (other than both positive axes) with labeling on the outside of the axes, use four-argument form of the tag with arguments corresponding to missing axes set to 0.

bounds specifies lengths of the axes but does not draw the axes (i.e., axes are invisible); x and y values are in dots relative to the -gorigin- location

bounds NEGATIVE X,NEGATIVE Y,POSITIVE X,POSITIVE Y
bounds POSITIVE X,POSITIVE Y

scalex specifies the maximum value and the value at the origin on the x axis; reset with each new -axes- or -bounds-; remains in effect across main unit boundaries until reset

scalex MAXIMUM VALUE OF X,*VALUE OF X AT ORIGIN*
(value at origin, if omitted, is 0)

scaley same options as -scalex- but for the y axis

scaley MAXIMUM VALUE OF Y,*VALUE OF Y AT ORIGIN*
(value at origin, if omitted, is 0)

NOTE: All subsequent graphing commands are in appropriate scaled units.

labelx specifies mark intervals, draws marks, and labels the x axis (with the specified number of places to the left and right of the decimal point)

labelx MAJOR INTERVAL,MINOR INTERVAL,MARKSIZE,LEFT,RIGHT

labely same options as -labelx- but for the y axis

labely MAJOR INTERVAL,MINOR INTERVAL,MARKSIZE,LEFT,RIGHT

markx specifies mark intervals; draws marks on the x axis with no labels

markx MAJOR INTERVAL,*MINOR INTERVAL,MARKSIZE*

marky same options as -markx- but for the y axis

marky MAJOR INTERVAL,*MINOR INTERVAL,MARKSIZE*

NOTE: With -labelx-, -labely-, -markx-, -marky-:
MINOR INTERVAL = 0 no minor marks (or can be omitted with -markx-, -marky- if MARKSIZE is also omitted)
MARKSIZE = 0 normal marks (or can be omitted with -markx-, -marky-)
= 1 major marks extending to the bounds of the graph
= 2 all marks extending to the bounds of the graph

gat similar to -at- but specifies the screen location relative to the -gorigin- location and in scaled units

gat X-LOCATION,Y-LOCATION
gat (B) (equivalent to -gat 0,0-, i.e., the current -gorigin- location)

gatnm similar to -gat- but does not reset the left margin (see -atnm-)

gatnm X-LOCATION,Y-LOCATION

gdot draws a dot at the specified position relative to the -gorigin- location and in scaled units

gdot X-LOCATION,Y-LOCATION

gdraw like -draw- but relative to the -gorigin- location and in scaled units; after execution "zwherex", "zwherey" are set to the last point plotted

for example:

gdraw X1,Y1;X2,Y2 (draws a line on the graph)

gbox same options as -box- but draws a rectangle relative to the -gorigin- location; affected by preceding -scalex- and -scaley- (see -box-)

```
gbox    CORNER X,CORNER Y;OPPOSITE CORNER X,OPPOSITE CORNER Y;
        DOTS THICK
gbox    ;CORNER X,CORNER Y;DOTS THICK   (draws a box with opposite
           corner at current "zwherex", "zwherey")
```

gfill similar to -fill- but fills (-mode write- or -mode rewrite-) or erases (-mode erase- or -mode inverse-) a rectangle relative to the -gorigin- location; affected by preceding -scalex- and -scaley-; not affected by -size- or -rotate- (see -fill-)

```
gfill   CORNER LOCATION;OPPOSITE CORNER LOCATION
```

gcircle same options as -circle- but is affected by preceding -scalex- and -scaley-; draws an ellipse if the -scalex- and -scaley- settings are different (see -circle-)

```
gcircle RADIUS IN DOTS,START ANGLE,END ANGLE
           (specify basic radius before affected by -scalex-, -scaley-)
```

gvector same options as -vector- except draws vector symbol relative to the -gorigin- location and in scaled units (see -vector-)

```
gvector XTAIL,YTAIL;XHEAD,YHEAD;SIZE
gvector ;XHEAD,YHEAD;SIZE      (tail at "zwherex", "zwherey")
```

vbar draws a vertical bar at the specified location relative to the -gorigin- location and in scaled units

```
vbar    X-LOCATION,HEIGHT
```

hbar draws a horizontal bar at the specified location relative to the -gorigin- location and in scaled units

```
hbar    LENGTH,Y-LOCATION
```

Special display

tabset specifies 10 tabulator settings for use when the TAB key is pressed at an arrow; each setting is an 8-bit byte which gives the horizontal coarse-grid position on the screen; settings remain in effect until reset by another -tabset- command

```
tabset  STARTING 8-BIT VAR CONTAINING TAB SETTINGS
tabset  (B)   (clears previous -tabset- settings)
```

char permits specification of specially designed characters for display

```
char    NAME,COL1,COL2,COL3,COL4,COL5,COL6,COL7,COL8
char    NAME,STARTING 16-BIT VAR CONTAINING COLUMN PATTERNS
```

Note: In the 9-argument form, COL1 through COL8 specify which of the 16 dots are lit in each of the 8 columns of the character. In the 2-argument form, STARTING VAR is the first of 8 consecutive 16-bit variables, each specifying the dots in each of the 8 columns, as in the 9-argument form.
NAME can be a number or a defined constant. NAME represents the alternate font memory location and must be in the range 32 to 126 <u>or</u> 160 to 254.

plot displays the contents of the specified alternate font memory location

```
plot    EXPR   (EXPR gives the alternate font memory location)
plot    NAME   (NAME is a defined numerical constant which is the
               alternate font memory location)
```

Note: Memory location must be in the range 32 to 126 <u>or</u> 160 to 254.

charset loads the specified character set into the station's memory

```
charset CHARSET NAME
charset (STARTING VAR)  (variable name must be enclosed in parentheses)
```

Note: zreturn = -1 if the charset is loaded successfully
= 0 if the charset is not found
= +1 if an error occurs in reading the disk

getchar copies the pattern in the specified alternate font memory location into the specified buffer (8 consecutive 16-bit integer variables <u>or</u> 16 consecutive 8-bit integer variables); one column of the pattern is stored in each 16 bits

```
getchar NAME,STARTING VAR
```

Note: NAME can be a number or a defined constant.

micro loads the specified microtable into the station's memory

```
micro   MICROTABLE NAME
micro   (STARTING VAR)  (variable name must be enclosed in parentheses)
micro   (B)   (release memory space used by the microtable)

Note:  zreturn = -1  if the microtable is loaded successfully
               =  0  if the microtable is not found
               = +1  if an error occurs in reading the disk
```

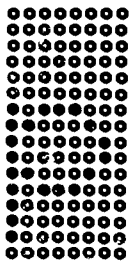
Color display

These features are available only on a station with color display.

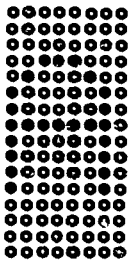
Definitions used in this subsection on color display:

primary colors: "zred", "zgreen", "zblue"
secondary colors: "zyellow" ("zred" + "zgreen")
"zcyan" ("zgreen" + "zblue")
"zmgenta" ("zred" + "zblue")
"zwhite": contains all primary colors ("zred" + "zgreen" + "zblue")
"zblack": contains no colors
character grid: the matrix of dots comprising the figure (1 bits or "•" dots) and field (0 bits or "o" dots) of a character

character "p"



character "A"



screen color: color of the screen outside the character grid (figure + field)
FOREGRND: foreground color
BACKGRND: background color
MASK: mask color; displays color component(s) common to the mask color and any specified color (foreground, background, and screen); e.g., if mask color is magenta and foreground color is cyan, display color is blue; if mask color is white, display colors are the same as the specified colors

color specifies colors for display

color FOREGRND,BACKGRND,MASK

Note: Colors are initially set to: -color zcyan,zblack,zwhite- .
A blank argument indicates color setting is unchanged. All three arguments cannot be blank simultaneously.
A full-screen erase fills the screen with the masked background color.
Screen color is initially black.

NOTE: Colors displayed by commands -erase- and -mode- are affected by MASK.

erase erases all or part of the color display

```
erase   (B)   (fills the screen with the masked background color)
erase   NUM CHARACTERS   (fills a rectangle NUM CHARACTERS long and
           1 character high with the masked background color)
erase   NUM CHARACTERS,NUM LINES   (fills a rectangle NUM CHARACTERS
           long and NUM LINES high with the masked background color)
```

mode specifies display mode

```
mode    write     (figure in the masked foreground color against the
                  masked screen color)
mode    erase     (figure in the masked background color against the
                  masked screen color)
mode    rewrite   (figure in the masked foreground color and field
                  in the masked background color against the masked
                  screen color)
mode    inverse   (figure in the masked background color and field
                  in the masked foreground color against the masked
                  screen color)
```

Non-screen presentation

enable allows input from the touch panel and from external devices; disabled at each main unit

```
enable  touch
enable  ext
enable  touch,ext   (can combine tags)
```

Note: At the end of a unit containing -enable touch-, a touch input is equivalent to pressing NEXT.
The touch panel is disabled at each main unit.

disable prevents input from any device except the keyset; this is the normal state of the station; the touch panel is disabled automatically at the beginning of each main unit

```
disable touch
disable ext
disable touch,ext
```

xout sends data (in 8-bit bytes) contained in the specified variables to an external device; data is read starting with the left-most byte

```
xout    DEVICE NUMBER,STARTING VAR,NUM 8-BIT BYTES TO SEND
```

xin collects data (in 8-bit bytes) from an external device and stores it in the specified variables starting at the left-most 8-bit byte

```
xin     DEVICE NUMBER,STARTING VAR,NUM 8-BIT BYTES TO STORE
```

beep (no tag) rings the sound device on the station

intrupt specifies a unit to execute (via -do-) when an interrupt is received from an external device

```
intrupt UNIT NAME
intrupt EXPR,NAMEM,NAME0,x,NAME2,q  (example of conditional form;
            argument q clears setting; argument x leaves setting
            unchanged)
```

Note: External interrupts are automatically disabled when the interrupt-handling unit is executed. The last command of this unit should be -enable ext- to re-enable external interrupts.

System variables for presenting

zdevice gives information on the most recent input to the station
= 0 if most recent input is from the keyset
= 1 if most recent input is from the touch panel
= 2 through 7 if most recent input is from an external device

zmode = -1 with -mode erase-
= 0 with -mode rewrite-
= 1 with -mode write-
= 2 with -mode inverse-

zretinf set by execution of -text-, -textn-; contains the number of lines of text displayed

zwherex current value of the fine-grid x location

zwherey current value of the fine-grid y location

zxmin fine-grid x location of the corner closest to the screen origin of the rectangle established by the -window- command or the -fine- command

zymin fine-grid y location of the corner closest to the screen origin of the rectangle established by the -window- command or the -fine- command

zxmax fine-grid x location of the corner furthest from the screen origin of the rectangle established by the -window- command or the -fine- command

zymax fine-grid y location of the corner furthest from the screen origin of the rectangle established by the -window- command or the -fine- command

zblack
zred
zgreen
zyellow
zblue
zmagenta
zcyan
zwhite
} available on color terminals

Additional notes on PRESENTING

Additional notes on PRESENTING

Additional notes on PRESENTING

ROUTING

R

R

Router lesson and curriculum information

router declares that the specified lesson is the router lesson in use

```
router  NAME
router  (STARTING VAR)  (variable name must be enclosed in parentheses)
```

lesson sets the system variable "zldone" to indicate whether a lesson is considered complete

```
lesson  complete     (sets "zldone" to -1)
lesson  incomplete   (sets "zldone" to 0)
lesson  no end       (sets "zldone" to +1)
lesson  EXPR,complete,incomplete,x,no end   (example of conditional
           form; argument x leaves "zldone" unchanged)
```

score places the value of the tag, rounded to the nearest integer, into the system variable "zscore"

```
score   EXPR   (value from 0 to 100)
score   (B)  or  score   NEGATIVE VALUE   (sets "zscore" to -1)
```

status reads the status area in memory and stores a value in the specified buffer or writes into the status area and changes its value to the value contained in the specified buffer; status area has length of 64 8-bit bytes

```
status  read;STARTING VAR,NUM 8-BIT BYTES    (reads status and stores
                                              its value in the buffer)
status  write;STARTING VAR,NUM 8-BIT BYTES   (changes the status to
                                              the value in the buffer)
```

Note: The status buffer is preserved during -jumpout- from an instructional lesson to the router lesson. The router can then store the status information on disk and recover it later to establish a status upon re-entering the lesson.

System variables for routing

zldone = -1 if the user has encountered -lesson complete-
= 0 if the user has encountered -lesson incomplete-
= +1 if the user has encountered -lesson no end-

zscore rounded value of the tag of the -score- command (value from 0 to 100); can be set to -1 with -score (B)- or -score NEGATIVE VALUE-

zrouten indicates entry conditions to the router lesson:
= 0 if this is the first entry to the router lesson
= 1 if this entry to the router is via -jumpout-
= 2 if the router is returned to when the end of the instructional lesson is reached
= 3 if the router is returned to when the instructional lesson is terminated by STOP1 keypress
= 4 if the router is returned to when an execution error occurs in the instructional lesson

zstatl length of status area in memory; 64 8-bit bytes

Additional notes on ROUTING

Additional notes on ROUTING

SEQUENCING

S

S

Naming a unit

unit names and initiates a section of a lesson (called a unit) which can be referred to by other sequencing commands

```
unit    NAME   (maximum of 8 characters in NAME)
unit    NAME(VVAR1,VVAR2,VVAR3;AVAR1,AVAR2)   (form with "value"
           arguments and "address" arguments; value arguments can be
           constants or variables; address arguments must be variables)
unit    NAME(VVAR1,VVAR2)          (value arguments only)
unit    NAME(;AVAR1,AVAR2,AVAR3)   (address arguments only)
```

Note: Maximum length of a condensed unit is about 3000 8-bit bytes.
No unit can be named "q" or "x".
A maximum of 10 arguments total can be accepted by a unit.

Automatic sequencing

NOTE: Commands -do-, -goto-, -jumpn-, and -jump- can have the conditional form:

```
goto    EXPR,NAMEM,NAME0,NAME1,x,NAME3,q
do      EXPR,NAMEM,NAME0,x,q,NAME3
```

Argument x is equivalent to absence of the command; argument q is equivalent to a branch to an empty unit. Special case is -do q-, which is equivalent to -goto q-. Argument q is not valid with -jump-.

These commands can pass up to 10 values to a unit with arguments, e.g.:

```
goto    NAME(VALUE1,VALUE2,,VALUE4)  (values can be expressions)
:
unit    NAME(VAR1,VAR2,VAR3,VAR4)    (VAR3 is unchanged)
```

or

```
do      NAME(VALUE1,VALUE2;VAR1,VAR2,VAR3)   (only -do- can pass
                                              address arguments)
:
unit    NAME(VVAR1,VVAR2;AVAR1,AVAR2,AVAR3)
```

examples of passing arguments:

passing value arguments

```
unit    give
        f:value
calc    value←sin(65°)
goto    take(value)   $$ can also pass the function itself
**
unit    take(pass)
        f:pass
* argument defined in local define set--same type as passed parameter
show    pass
```

passing value arguments and address arguments

```
unit    give
        f:result
do      take(25;result)
show    result
* value argument--value or variable; address argument--variable only
**
unit    take(val;addr)
        f:addr
        i,8:val
* defined arguments have same types as passed parameters
calc    addr←log(val)  $$ "result" in unit "give" is set to same value
```

more examples on the next page

passing an array of specified length

```
unit    give
        i,8:array(10)
zero    array(1),10        $$ zero array before packing information
pack    array(1),$sin(65°)
do      take(10;array)     $$ pass entire array as address argument
**
unit    take(len;fun)
        i,8:len,fun(10)
        f:result
compute result,fun(1),len
show    result
```

passing an array of length determined by program

```
unit    give
        i,8:array(10),numchr
pack    array(1),numchr$sin(65°)
do      take(numchr;array)
**
unit    take(len;fun)
        i,8:len,fun(*)
        f:result
compute result,fun(1),len
show    result
```

do — causes execution of the specified unit, without screen erase or change of main unit; returns to the original unit to execute commands following -do-

```
do      UNIT NAME
```

goto — causes execution of the specified unit, without changing main unit and without main-unit initializations (including screen erase); does not return to the original unit to execute commands following -goto-; does not clear the -do- stack

```
goto    UNIT NAME
```

jumpn — jumps to the specified unit but does not do any initializations, such as main unit, screen erase, etc.; clears the -do- stack

```
jumpn   UNIT NAME
```

jump — causes execution of the specified unit, with a full-screen erase (unless the erase is prevented: see -inhibit erase-) and change of main unit; performs initializations associated with entering a main unit; does not alter base-unit setting

```
jump    UNIT NAME
```

imain specifies a unit to execute at the start of every main unit in the lesson; later occurrence of the command overrides an earlier setting; equivalent to -do- at the beginning of each main unit

```
imain   UNIT NAME
imain   q   (turns off -imain- setting for remainder of lesson or
            until reset)
imain   EXPR,NAMEM,NAME0,x,NAME2,q   (example of conditional form;
           argument q clears setting; argument x leaves setting
           unchanged)
```

NOTE: The following two commands (-branch-, -doto-) permit branching or looping within a unit to a line with a statement label. The line with the statement label must be in the same unit as -branch- or -doto-. The statement label must start with a digit and can contain a maximum of 7 characters, consisting of digits and letters only.

branch causes a branch to the line with the specified statement label

for example:

```
5a      VAR←EXPR
do      someu
write   some message
.
:
branch  EXPR,5a,x     (argument x causes fall-through to the next line
                      in the unit)
```

doto causes iterative execution of lines of the program between -doto- and the line with the specified statement label while changing a counter; the labeled line must have a blank tag

for example:

```
doto    2sync,VAR←INITIAL EXPR,FINAL EXPR,STEPSIZE EXPR
do      someu
write   some message
.
:
2sync   (B)
```

Note: Stepsize, if omitted, is +1. Stepsize can be negative. The loop variable must be an integer variable. Its value is undefined after completion of the loop.

NOTE: The following four commands (-if-, -elseif-, -else-, and -endif-) permit branching within a unit. Logical value of an expression is "true" if its rounded value is -1 and "false" if its rounded value is 0. (In general, a value < 0 is "true" and a value ≥ 0 is "false".)

if — performs a branch based on the logical value of the tag expression; value of "true" causes fall-through to the next line; value of "false" causes branch to the next -elseif-, -else-, or -endif- at the same level; code following -if- must be indented (up to the next -elseif-, -else-, or -endif- at the same level) and marked with the "indent" symbol; range of -if- must be terminated by -endif- at the same level

```
if      LOGICAL EXPR
```

elseif — provides an alternative branch within the range of the preceding -if- at the same level; subsequent code follows same indenting rules as -if-

```
elseif  LOGICAL EXPR
```

else — (no tag) provides a branch if the logical value of the tag of the preceding -if- or -elseif- at the same level is "false"; subsequent code follows same indenting rules as -if-

endif — (no tag) marks the end of the range of the preceding -if- at the same level

NOTE: Following is an example demonstrating placement of these commands and the form of the "indent symbol". The indent symbol is a "." followed by at least one space.

```
if      a<4
.       write   first branch            $$ executed if a<4
.       calc    b¢3⁴                    $$ executed if a<4
elseif  a=4                             $$ executed if a≥4
.       write   second branch           $$ executed if a=4
.       do      someunit                $$ executed if a=4
else                                    $$ executed if a>4
.       write   default branch          $$ executed if a>4
.       if      a>6                     $$ executed if a>4
.       .       write   special branch  $$ executed if a>6
.       endif                           $$ end of range of -if  a>6-
endif                                   $$ end of range of -if  a<4-
```

NOTE: The following four commands (-loop-, -endloop-, -outloop-, and -reloop-) permit looping within a unit. Logical value of an expression is "true" if its rounded value is -1 and "false" if its rounded value is 0. (In general, a value < 0 is "true" and a value ≥ 0 is "false".)

loop — initiates a loop based on the logical value of the tag expression; value of "true" causes execution of subsequent commands in the loop; value of "false" causes execution of the first command after -endloop- at the same level of indentation as -loop-; code following -loop- must be indented (up to the next -outloop-, -reloop-, or -endloop- at the same level) and marked with the "indent" symbol; range of -loop- is marked by -endloop- at the same level

```
loop    LOGICAL EXPR   (blank tag is equivalent to "true" value)
```

endloop — (no tag) marks the end of a loop initiated by the previous -loop- command at the same level of indentation; causes a branch back to the previous -loop- command at the same level

outloop — based on the logical value of the tag, causes exit from the range of -loop- at the same level of indentation; value of "true" causes execution of the first command after -endloop- at the same level; value of "false" causes execution of subsequent commands within the loop, which follow the same indenting rules as -loop-

```
outloop LOGICAL EXPR   (blank tag is equivalent to "true" value)
```

reloop — based on the logical value of the tag expression, causes branch back to the previous -loop- command at the same level of indentation without terminating the loop; value of "true" causes branch to the previous -loop- at the same level; value of "false" causes execution of subsequent commands within the loop, which follow the same indenting rules as -loop-

```
reloop  LOGICAL EXPR   (blank tag is equivalent to "true" value)
```

NOTE: Following is an example demonstrating placement of these commands and the form of the "indent symbol". The indent symbol is a "." followed by at least one space.

```
loop    a<10
.       write   within loop          $$ executed if a<10
.       calc    a¢a-1                $$ executed if a<10
reloop  a≥5                          $$ executed if a<10
.       write   still within loop    $$ executed if a<5
.       do      someunit             $$ executed if a<5
outloop a<3                          $$ executed if a<5
.       write   still within loop    $$ executed if 3≤a<5
endloop
write   outside of loop              $$ executed if a≥10 or a<3
```

Key-initiated sequencing

NOTE: Commands -next- through -lablop- can have the conditional form, where argument x leaves the pointer unchanged, and argument q clears the pointer and renders the key inactive. The conditional expression is evaluated when the command is executed, not when the key is pressed. A full-screen erase is performed at the beginning of a main unit unless the unit is an "op" unit or the previous unit contained -inhibit erase-.

next, next1, back, back1, stop specifies the unit executed when the user presses the appropriate key (with -next- the user must be at the end of the unit before the specified unit is executed); the specified unit is a main unit

next UNIT NAME
back1 q (clears back1 pointer; disables SHIFT BACK key)

nextop, next1op, backop, back1op, stopop specifies the unit executed when the user presses the appropriate key; there is no full-screen erase and additional plotting is on the same display (on-the-page); the specified unit is a main unit

next1op UNIT NAME
backop q (clears the back pointer; disables BACK key)

help, help1, data, data1, lab, lab1 specifies the unit executed when the user presses the appropriate key; initiates a help sequence and sets the base pointer to the unit containing the help-type command (unless the base pointer is already set); the base unit is executed after the help sequence is completed or if the user presses BACK or BACK1; units in a help sequence are main units but not base units; a help sequence ends with a unit that does not contain a -next- command

help UNIT NAME
lab q (clears lab pointer; disables LAB key)

helpop, help1op, dataop, data1op, labop, lab1op specifies the unit executed when the user presses the appropriate key; there is no full-screen erase and additional plotting is on the same display (on-the-page); the unit executed is neither a main unit nor a base unit; after execution, control is returned to the unit containing -helpop- at the -arrow-, -pause-, or end-of-unit where the user pressed the key

helpop UNIT NAME
dataop q (clears the data pointer; disables the DATA key)

base clears the base pointer in order to alter help-type sequencing

base (B)

Pausing and timing

keylist (non-executable) forms a set of keys with the specified name for use with -pause-, -keytype-, and -jkey- commands

```
keylist NAME,KEY1,KEY2,KEY3,...    (from 2 to 7 characters in NAME)
keylist NAME,NAME1,NAME2,...       (keylists can be combined)
```

Note: System-defined keylists are:

alpha	(letters: a to z and A to Z)
numeric	(digits: 0 to 9)
funct	(function keys)
keyset	(any keyset input)
touch	(input from touch panel)
ext	(input from external device other than touch panel)
all	(input from keyset, touch panel, or external device)

In addition to alpha, numeric, and function keys, names given in the tables in the appendix can be used. These names can also be used with -press-, -jkey-, -pause-, -keytype-, and "zk(KEYNAME)".

pause delays execution of subsequent commands by the specified interval (and then presses TIMEUP key) or until one of the specified keys is pressed

```
pause   EXPR GIVING NUM SECONDS
pause   0   (causes no pause)
pause   (B)  or  pause   NEGATIVE VALUE   (interrupts processing until
           any key or allowed input comes in)
pause   keys=KEY1,KEY2,KEYLIST NAME,...   (interrupts processing until
           a specified key is pressed; all keynames are typed without
           quote marks and function keys are typed in lower case)
pause   NUM SECONDS,keys=KEY1,KEY2,KEYLIST NAME,...   (interrupts
           processing for the specified interval [and then presses the
           TIMEUP key] or until a specified key is pressed)
```

Note: If a function key oth than next, such as help, is specified and there is a preceding -help- command specifying a unit, this unit is executed rather than the command following the -pause-. If next is specified, the NEXT key just breaks the -pause-, even if there is a preceding -next- command.
The statements -pause keys=touch- and -pause keys=ext- set the appropriate -enable-; -pause keys=all- does not set -enable-.
Keycode of the input (including TIMEUP) is stored in "zkey".

keytype sets a variable according to the position in a list of the input by the user; if the input action is not listed, the variable is set to -1

keytype VAR,ARG0,ARG1,ARG2,...

arguments ARG0, ARG1, ARG2,... can be any of the following:

KEYNAME (any keyname; no quotation marks are used; function keys are in lower case)
KEYLIST NAME (name of a system-defined keylist or of a list set up by the -keylist- command)
ext (any external input)
touch(COARSE,*WIDTH IN CHARACTERS,HEIGHT IN LINES*)
touch(FINEX,FINEY,*WIDTH IN DOTS,HEIGHT IN DOTS*)
(COARSE or FINEX,FINEY is the screen position of the corner closest to the screen origin (as defined by the -fine- command) of a rectangle with specified width and height; width and height are optional and have value 1 if omitted)

Note: Up to 97 keys can be specified; keylists count as one key.

press puts the specified key into the student input buffer

press zk(KEYNAME) (e.g., press zk(b) <u>or</u> press zk(back) <u>or</u> press zk(super))
press "KEYNAME" (e.g., press "b" ; function keys not allowed)
press EXPR GIVING KEYCODE
press zk(touch)+X (touch input at screen coordinates X,Y;
press zk(touch)+Y both lines of code must be given)
press zk(ext)+D (input of data D from an external device)

getkey (no tag) reads the next key from the key buffer (which contains up to 12 keys pressed by the user or by -press-), removes the key from the buffer, and sets "zkey" to the value of the key (sets "zkey" to -1 if the buffer is empty)

clrkey (no tag) clears the key buffer

Lesson connections and sections

use (non-executable) inserts into the file being condensed the specified block(s) from the file specified in the tag of -use-; all contiguous blocks with the same name are taken

```
use     FILE NAME,BLOCK NAME
```

jumpout causes execution of the specified lesson

```
jumpout FILE NAME
jumpout FILE NAME(VALUE1,VALUE2)   (example of form with arguments;
           up to 10 values can be passed to the first unit in the
           lesson; first unit is specified with arguments)
jumpout (STARTING VAR)  (variable name must be enclosed in parentheses)
jumpout (STARTING VAR)(VALUE1,VALUE2)   (variable name, arguments)
jumpout q  or  jumpout (B)   (returns to the router lesson)
```

cstart (non-executable) (no tag) indicates subsequent code is to be condensed (used after a preceding -cstop-)

cstop (non-executable) (no tag) indicates subsequent code is not to be condensed; in effect up to the next -cstart-, if any

cstop* (non-executable) (no tag) indicates none of the subsequent code is to be condensed, independent of subsequent -cstart- commands

Lesson annotation and debugging

* indicates the statement on that line is a comment only and is to be ignored by the computer

```
*This is a comment.
```

$$ (not a command) when placed on the same line with a µTUTOR statement indicates that subsequent material on that line is a comment

```
COMMAND TAG   $$this is a comment
```

Note: Spaces preceding the $$ are discarded.

step allows a user to step through a lesson command by command; information about execution is displayed at the bottom of the screen

```
step    EXPR   (value=-1 turns step mode on;
               value=0 turns step mode off)
```

Signing off

finish specifies the unit which will be executed when the user leaves the lesson by pressing STOP1

```
finish UNIT NAME
finish  q   (clears -finish- setting)
finish  EXPR,NAMEM,NAME0,q,NAME2,x   (example of conditional form;
           argument q clears setting; argument x leaves setting
           unchanged)
```

protect prevents the lesson from being interrupted by STOP1 keypress; if STOP1 is pressed during execution of program code after protection is turned on, execution continues until the lesson ends or until protection is turned off; the finish unit (if any) is executed before the user leaves the lesson

```
protect EXPR   (value=-1 turns protection on;
               value=0 turns protection off)
```

System variables for sequencing

zargs — number of values passed at the previous execution of a unit with arguments or -jumpout- with arguments

zclock — number of seconds (to the nearest millisecond) since midnight (floating-point number)

zday — number of days between the current day and midnight Sunday, December 31, 1972 to the nearest 10^{-6} day (approximately .1 second) (floating-point number)

zenvir — = 1 if the terminal is running on a Cluster network
= 2 if the terminal is running independent of a Cluster network

zkey — contains information on the last input (updated after -arrow-, -pause-, -getkey-, and at the beginning of a main unit); counting from the left end of the 16-bit word (2# 0sssskkkkdddddddd):
1 bit: always 0
3 bits (sss): source of input: 000 keyset; 001 touch panel; 010 external device
4 bits (kkkk): set if source is keyset: 0000 displayable character; 0001 function key; 0010 Japanese function key
8 bits (dddddddd): ASCII code for character if source is keyset; value of data if source is external device; 0 if source is touch panel (see "ztouchx", "ztouchy")

znumpad — = -1 if the most recent input is from the number pad
= 0 otherwise (or if the keyset does not have a number pad)

zport — identification number of the port on the Cluster System

zreturn — set by many commands according to the results of execution; set by: -addfile-, -addname-, -charset-, -chgfile-, -chgname-, -compute-, -datain-, -dataout-, -delfile-, -delname-, -getfile-, -getname-, -putd-, -micro-, -names-, -reserve-, -release-, -setdir-, -setfile-, -setname-, -textn-

ztouchx — fine-grid x location of the center of the touch square touched (set to 0 if source of input is not touch panel)

ztouchy — fine-grid y location of the center of the touch square touched (set to 0 if source of input is not touch panel)

```
ztstype  gives information on the terminal's subtype
         if the value of "zttype" is 34:
         = 0  if the terminal is an IST-2
         = 1  if the terminal is an IST-3
         = 2  if the terminal is a Viking

         if the value of "zttype" is 42:
         = 0  if the terminal is an NEC-8801
         = 1  if the terminal is an NEC-9801
         = 2  if the terminal is an NEC-9801 running under local operating
              system using floppy disks

zttype   gives information on the user's terminal type
         = 34  if the terminal is an IST or Viking (512x512 monochrome screen)
         = 42  if the terminal is an NEC (640x400 8-color screen)
```

Additional notes on SEQUENCING

Additional notes on SEQUENCING

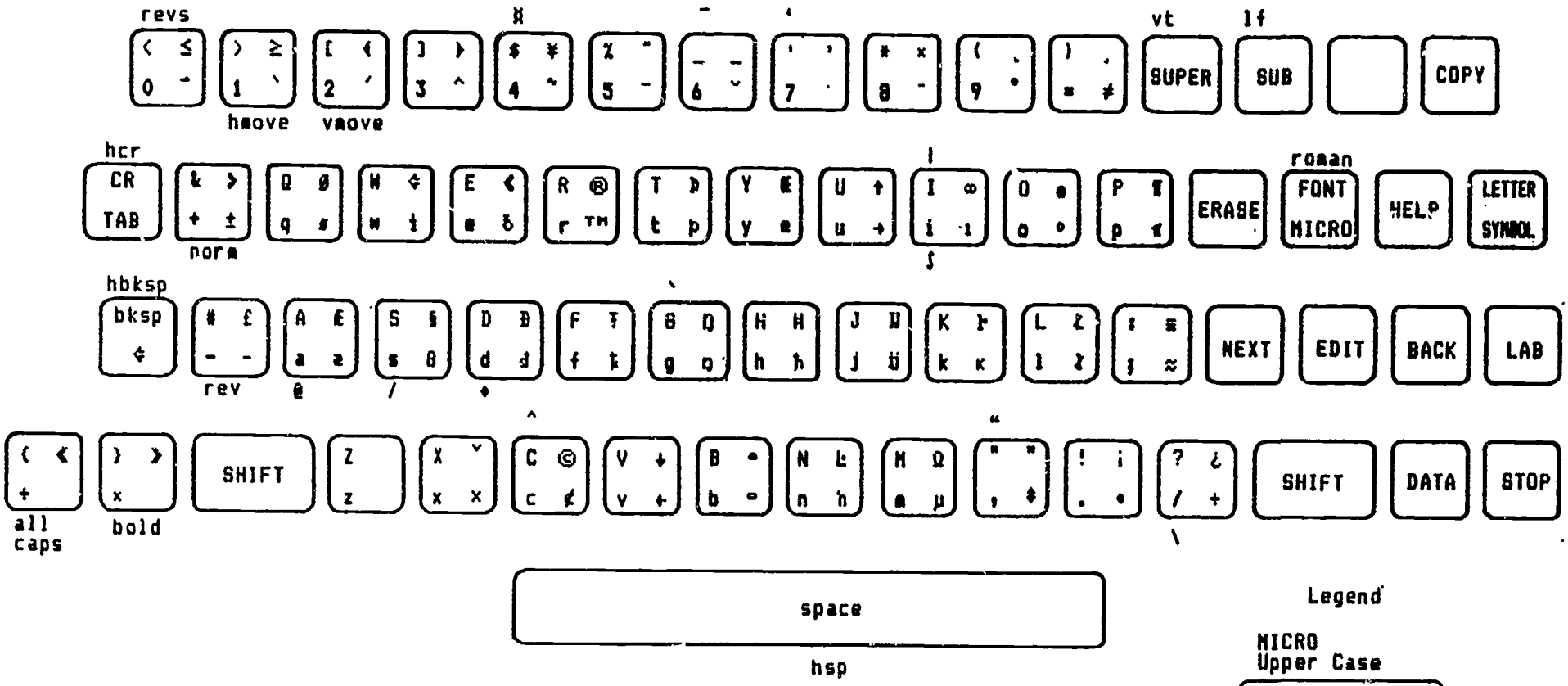
APPENDIX

A

A

Contents of the Appendix

IST Keyset



Legend

MICRO Upper Case

Upper Case	SYMBOL Upper Case
Lower Case	SYMBOL Lower Case

MICRO Lower Case

TDK Keyset

SFULL FULL | STD HALF | SKANJI KANJI | KATAKANA | HIRAGANA | ROMAN | TRANS | WORD | KANJID

vt SUPER | 1 ! ¡ ` hsove | 2 " ¨ ´ vsove | 3 # £ ^ | 4 $ ¥ ~ | 5 % ¯ | 6 & > ˘ | 7 ' ˙ | 8 (: | 9) ° | 0 \ / | - = ≠ – rev | ^ ~ ˛ | ¥ | TERM | FONT MICRO (roman) | LETTER SYMBOL | ↑ | PRINT | ESC | CTRL

lf SUB | Q @ q ø | W ÷ w ł | E ¢ e ß | R ® r ™ | T ⅜ t þ | Y ⅝ y ø | U ↑ u ↓ | I ∞ i ı | O ○ o ° | P ¶ p ¤ | ` ' ` ¤ | { « [‹ | ERASE | BKSP hbksp | EDIT | COPY | 7 | 8 | 9 | + all caps

hcr CR TAB | A Æ a æ | S § s ß | D ⅜ d đ | F ⅞ f ⅛ | G © g ŋ | H ⅓ h ħ | J ⅔ j ij | K ⅛ k ĸ | L ⅜ l ł | + ± ; ≈ (more) | * × : ≡ |) »] › | NEXT | LAB | BACK | 4 | 5 | 6 | × bold

CAPS LOCK | SHIFT | Z z | X ˇ x × | C © c ¢ | V ↓ v ↓ | B ⅝ b ⅝ | N ⅛ n ħ | M Ω m µ | < ≤ , ¸ (revs) | > ≥ . • | ? ¿ / ÷ | _ – | SHIFT | DATA | HELP | APS | 1 | 2 | 3 | - rev

space hsp | STOP | 0 | . | NEXT | + more

Legend

MICRO Upper Case

Upper Case	SYMBOL Upper Case
Lower Case	SYMBOL Lower Case

MICRO Lower Case

Control Codes and Character Codes

Control Codes				Character Codes													
C	H	C	H	C	H	C	H	C	H	C	H	C	H	C	H	C	H
				space	20	0	30	@	40	P	50	`	60	p	70
				!	21	1	31	A	41	Q	51	a	61	q	71
				"	22	2	32	B	42	R	52	b	62	r	72
				#	23	3	33	C	43	S	53	c	63	s	73
				$	24	4	34	D	44	T	54	d	64	t	74
				%	25	5	35	E	45	U	55	e	65	u	75
				&	26	6	36	F	46	V	56	f	66	v	76
				'	27	7	37	G	47	W	57	g	67	w	77
bksp	08			(	28	8	38	H	48	X	58	h	68	x	78
		kata	19	)	29	9	39	I	49	Y	59	i	69	y	79
lf	0a			*	2a	:	3a	J	4a	Z	5a	j	6a	z	7a
vt	0b	escape	1b	+	2b	;	3b	K	4b	[	5b	k	6b	{	7b
				,	2c	<	3c	L	4c	\	5c	l	6c	|	7c
cr	0d			-	2d	=	3d	M	4d	]	5d	m	6d	}	7d
hira	0e			.	2e	>	3e	N	4e	^	5e	n	6e	~	7e
roman	0f			/	2f	?	3f	O	4f	_	5f	o	6f		

Control Codes				Character Codes													
C	H	C	H	C	H	C	H	C	H	C	H	C	H	C	H	C	H
rev	80	full	90			°	b0	‾	c0	‐	d0	Ω	e0	κ	f0
revs	81	kanji	91	¡	a1	±	b1	`	c1	∫	d1	Æ	e1	æ	f1
bold	82	hmove	92	¢	a2	◆	b2	´	c2	®	d2	Đ	e2	đ	f2
norm	83	vmove	93	£	a3	●	b3	^	c3	©	d3	ª	e3	ð	f3
sfull	84			⁄	a4	×	b4	~	c4	™	d4	Ħ	e4	ħ	f4
half	85			¥	a5	µ	b5	¯	c5	⅛	d5	‡	e5	ı	f5
std	86			ƒ	a6	¶	b6	˘	c6	≡	d6	Ĳ	e6	ĳ	f6
vert	87			§	a7	•	b7	˙	c7	≠	d7	Ł	e7	ł	f7
hbksp	88			¤	a8	÷	b8	¨	c8	≤	d8	Ŀ	e8	ŀ	f8
hsp	89			'	a9	'	b9	/	c9	≥	d9	Ø	e9	ø	f9
sub1	8a			"	aa	"	ba	˚	ca	⅜	da	Œ	ea	œ	fa
super1	8b			«	ab	»	bb	¸	cb	⅝	db	º	eb	ß	fb
hcr	8c			←	ac			˝	cc	⅞	dc	Þ	ec	þ	fc
nl	8d			↑	ad	¼	bd	˛	cd	×	dd	Ŧ	ed	ŧ	fd
skanji	8e			→	ae			ˇ	ce	∞	de	Ŋ	ee	ŋ	fe
font	8f			↓	af	¿	bf		cf	∝	df	ŉ	ef		

C ≡ control or character
H ≡ ASCII code in hexadecimal notation

The system variable "zkey" contains the code value of the previous input.
The function "zk" gives the values in these tables: zk(hsp)=89h; zk(next)=101h.
With some characters, special names must be used with "zk":
textsep (for ⁄); lembed (for «); rembed (for »); comma (for ,).

Explanation of Control Names

[Names in dark type indicate features available only on Japanese keysets.]

bksp	backspace
lf	line feed (move down one line; distance moved is determined by the size of the characters in use)
vt	vertical tab (move up one line; distance moved is determined by the size of the characters in use)
cr	carriage return with line feed whose height is determined by the size of the characters in use
hira	hiragana characters
roman	standard roman alphabet
kata	katakana characters
escape	escape code
rev	writing right to left in alternate font from current screen location
revs	writing right to left in alternate font from right edge of window
bold	bold writing
norm	normal writing (left to right, standard size)
sfull	short full-size characters (16 dots × 16 dots)
half	half-size characters (8 dots × 24 dots)
std	standard-size characters (8 dots × 16 dots)
vert	vertical writing (upward)
hbksp	half backspace
hsp	half space
subl	locked subscript (characters are written 5 dots below line of text)
superl	locked superscript (characters are written 5 dots above line of text)
hcr	half carriage return
nl	new line; similar to cr but restores display conditions (bold, font, etc.) present at the beginning of the text (beginning of the -write- command); automatically inserted by the condensor for each new line of text
skanji	short kanji characters (16 dots × 16 dots)
font	alternate font characters
full	full-size characters (16 dots × 24 dots)
kanji	kanji characters (16 dots × 24 dots)
hmove	horizontal movement of 1 dot (right if writing is left to right, left if writing is right to left)
vmove	vertical movement of 1 dot upward

Characters and controls can be embedded in text, for example:
‹cr›, ‹roman›, ‹font›, ‹hira›, ‹kata›, ‹skanji›, ‹bold›, ‹norm›

Codes for Function Keys

C	H	C	H	C (Japanese)	H
timeup	100				
next	101	next1	111	trans	201
back	102	back1	112	wordd	202
help	103	help1	113	kanjid	203
lab	104	lab1	114		
data	105	data1	115		
stop	106	stop1	116		
ans	107				
term	108				
erase	109	erase1	119		
edit	10a	edit1	11a		
copy	10b	copy1	11b		
symbol	10c	letter	11c		
print	10d	sub	11d		
tab	10e	super	11e		
micro	10f				

zk(touch) = 1000h
zk(ext) = 2000h

Characters and SYMBOL Characters

C	S+C	C	S+C	C	S+C	C	S+C	C	S+C	C	S+C
		0	¯	@	¤	P	¶	`	´	p	¥
!	¡	1	`	A	Æ	Q	Ø	a	æ	q	ß
"	"	2	´	B	ª	R	®	b	ª	r	™
#	£	3	^	C	©	S	§	c	¢	s	ß
$	¥	4	~	D	Ð	T	Þ	d	ð	t	þ
%	¯	5	¯	E	‹	U	↑	e	δ	u	→
&	»	6	˘	F	¢	V	↓	f	¢	v	←
'	'	7	·	G	Ŋ	W	¢	g	ŋ	w	¿
(	¸	8	¨	H	H	X	ˇ	h	ħ	x	×
)	¸	9	°	I	∞	Y	Æ	i	ı	y	æ
*	×	:	≡	J	Ĳ	Z		j	ĳ	z	
+	±	;	≈	K	ĸ	[	‹	k	ĸ	{	«
,	‡	<	≤	L	Ł	\	/	l	ł	|	∫
-	-	=	≠	M	Ω	]	»	m	µ	}	»
.	•	>	≥	N	ŉ	^	♦	n	ŉ	~	"
/	÷	?	¿	O	●	_	_	o	°		

C ≡ character with ASCII code between 32 and 127
S+C ≡ SYMBOL+C ≡ character with ASCII code between 160 and 255
≡ [press SYMBOL key and release; press character key {C}]

See table below **Characters and MICRO Characters** for exceptions with the IST.

Characters and MICRO Characters

C	M+C	C	M+C
a	@	$	¤
/	\	s	/
C	^	d	♦
G	`	'	´
I	|	i	∫
-	~	"	"

C ≡ character
M+C ≡ MICRO+C ≡ character not accessible directly or with SYMBOL on the IST keyset

MICRO Characters and Display Functions

C	M+C	C	M+C
space	hsp	x	bold
bksp	hbksp	+	norm
sub1	lf	-	rev
super1	vt	<	revs
font	roman	1	hmove
cr	hcr	2	vmove

C ≡ character or control
M+C ≡ MICRO+C ≡ display function not marked on the keyset

M+C ≡ [press MICRO key and release; press character key or control key {C}]

Note: These MICRO characters may not be available if a user-defined microtable is in effect.

Definition of Hexadecimal Symbols

decimal	binary	hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	a
11	1011	b
12	1100	c
13	1101	d
14	1110	e
15	1111	f

Conversion between Decimal (D) and Hexadecimal (H) Numbers

D	H	D	H	D	H	D	H	D	H	D	H	D	H	D	H
0	00	32	20	64	40	96	60	128	80	160	a0	192	c0	224	e0
1	01	33	21	65	41	97	61	129	81	161	a1	193	c1	225	e1
2	02	34	22	66	42	98	62	130	82	162	a2	194	c2	226	e2
3	03	35	23	67	43	99	63	131	83	163	a3	195	c3	227	e3
4	04	36	24	68	44	100	64	132	84	164	a4	196	c4	228	e4
5	05	37	25	69	45	101	65	133	85	165	a5	197	c5	229	e5
6	06	38	26	70	46	102	66	134	86	166	a6	198	c6	230	e6
7	07	39	27	71	47	103	67	135	87	167	a7	199	c7	231	e7
8	08	40	28	72	48	104	68	136	88	168	a8	200	c8	232	e8
9	09	41	29	73	49	105	69	137	89	169	a9	201	c9	233	e9
10	0a	42	2a	74	4a	106	6a	138	8a	170	aa	202	ca	234	ea
11	0b	43	2b	75	4b	107	6b	139	8b	171	ab	203	cb	235	eb
12	0c	44	2c	76	4c	108	6c	140	8c	172	ac	204	cc	236	ec
13	0d	45	2d	77	4d	109	6d	141	8d	173	ad	205	cd	237	ed
14	0e	46	2e	78	4e	110	6e	142	8e	174	ae	206	ce	238	ee
15	0f	47	2f	79	4f	111	6f	143	8f	175	af	207	cf	239	ef
16	10	48	30	80	50	112	70	144	90	176	b0	208	d0	240	f0
17	11	49	31	81	51	113	71	145	91	177	b1	209	d1	241	f1
18	12	50	32	82	52	114	72	146	92	178	b2	210	d2	242	f2
19	13	51	33	83	53	115	73	147	93	179	b3	211	d3	243	f3
20	14	52	34	84	54	116	74	148	94	180	b4	212	d4	244	f4
21	15	53	35	85	55	117	75	149	95	181	b5	213	d5	245	f5
22	16	54	36	86	56	118	76	150	96	182	b6	214	d6	246	f6
23	17	55	37	87	57	119	77	151	97	183	b7	215	d7	247	f7
24	18	56	38	88	58	120	78	152	98	184	b8	216	d8	248	f8
25	19	57	39	89	59	121	79	153	99	185	b9	217	d9	249	f9
26	1a	58	3a	90	5a	122	7a	154	9a	186	ba	218	da	250	fa
27	1b	59	3b	91	5b	123	7b	155	9b	187	bb	219	db	251	fb
28	1c	60	3c	92	5c	124	7c	156	9c	188	bc	220	dc	252	fc
29	1d	61	3d	93	5d	125	7d	157	9d	189	bd	221	dd	253	fd
30	1e	62	3e	94	5e	126	7e	158	9e	190	be	222	de	254	fe
31	1f	63	3f	95	5f	127	7f	159	9f	191	bf	223	df	255	ff

Powers of 2

n	2^n	n	2^n
0	1	30	1 073 741 824
1	2	31	2 147 483 648
2	4	32	4 294 967 296
3	8	33	8 589 934 592
4	16	34	17 179 869 184
5	32	35	34 359 738 368
6	64	36	68 719 476 736
7	128	37	137 438 953 472
8	256	38	274 877 906 944
9	512	39	549 755 813 888
10	1 024	40	1 099 511 627 776
11	2 048	41	2 199 023 255 552
12	4 096	42	4 398 046 511 104
13	8 192	43	8 796 093 022 208
14	16 384	44	17 592 186 044 416
15	32 768	45	35 184 372 088 832
16	65 536	46	70 368 744 177 664
17	131 072	47	140 737 488 355 328
18	262 144	48	281 474 976 710 656
19	524 288	49	562 949 953 421 312
20	1 048 576	50	1 125 899 906 842 624
21	2 097 152	51	2 251 799 813 685 248
22	4 194 304	52	4 503 599 627 370 496
23	8 388 608	53	9 007 199 254 740 992
24	16 777 216	54	18 014 398 509 481 984
25	33 554 432	55	36 028 797 018 963 968
26	67 108 864	56	72 057 594 037 927 936
27	134 217 728	57	144 115 188 075 855 872
28	268 435 456	58	288 230 376 151 711 744
29	536 870 912	59	576 460 752 303 423 488

Additional notes on Keysets and Characters

INDEX

I

Alphabetical index to system variables

System variables can be used wherever expressions are accepted.

Alphabetical list of system functions and operations

These functions are described on pages C4 to C6.

Alphabetical index to commands and related directives

Alphabetical index to commands and related directives (cont.)

C

F

J

P

R

S

A

I

C CALCULATING

F FILE OPERATIONS

J JUDGING

P PRESENTING

R ROUTING

S SEQUENCING

A APPENDIX

I INDEX