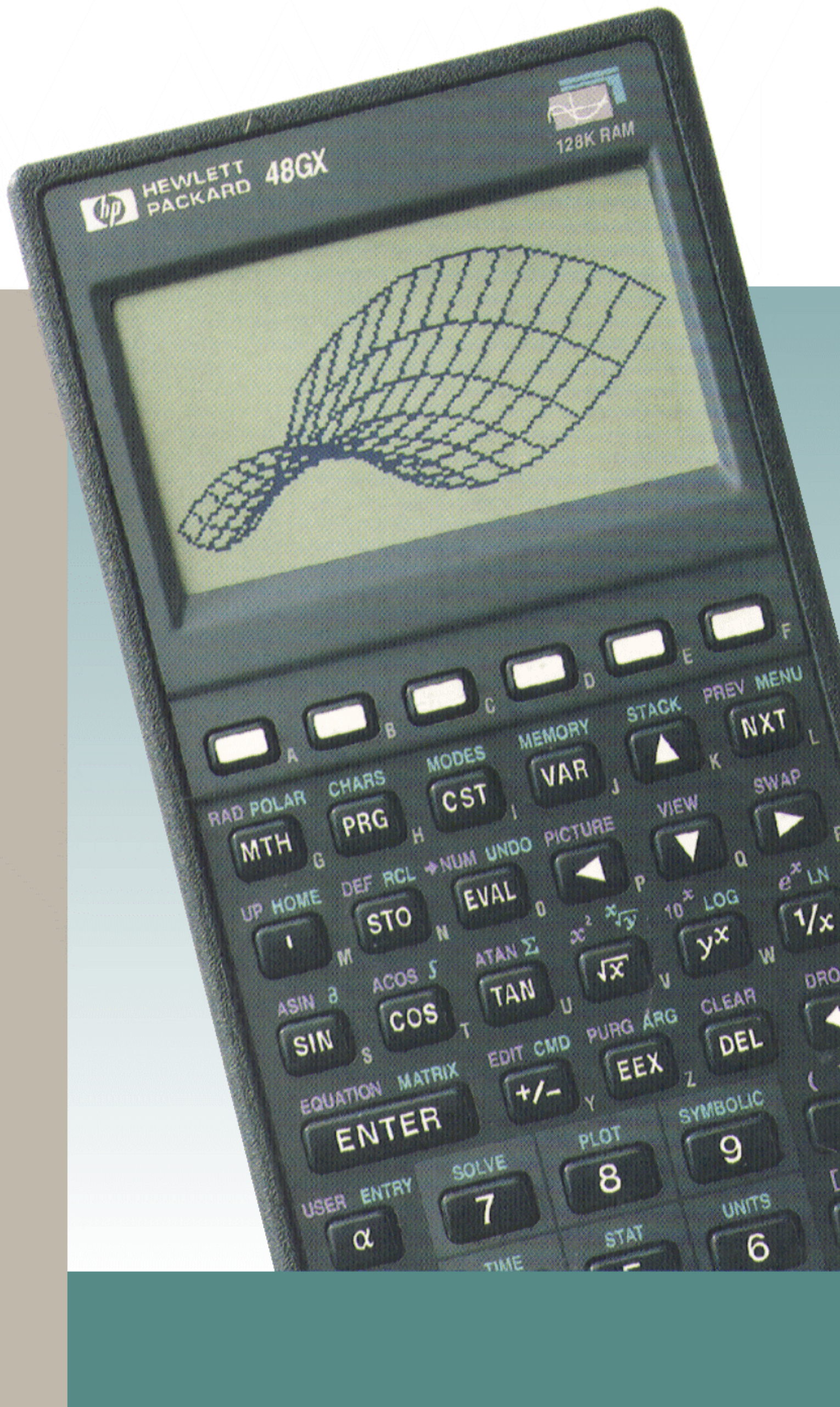
HEWLETT PACKARD

HP 48G Series

Advanced User's Reference Manual



HP 48G Series Advanced User's Reference Manual

HP Part No. 00048-90136
Printed in Singapore

Notice

This manual and any examples contained herein are provided "as is" and are subject to change without notice. **Hewlett-Packard Company makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.** Hewlett-Packard Co. shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual or the examples herein.

© Copyright Hewlett-Packard Company 1993. All rights reserved.
Reproduction, adaptation, or translation of this manual is prohibited without prior written permission of Hewlett-Packard Company, except as allowed under the copyright laws.

The programs that control this product are copyrighted and all rights are reserved. Reproduction, adaptation, or translation of those programs without prior written permission of Hewlett-Packard Co. is also prohibited.

© Trustees of Columbia University in the City of New York, 1989. Permission is granted to any individual or institution to use, copy, or redistribute Kermit software so long as it is not sold for profit, provided this notice is retained.

Hewlett-Packard Company
Corvallis Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

Acknowledgements

Hewlett-Packard gratefully acknowledges the members of the Education Advisory Committee (Dr. Thomas Dick, Dr. Lynn Garner, Dr. John Kenelly, Dr. Don LaTorre, Dr. Jerold Mathews, and Dr. Gil Proctor) for their assistance in the development of this product. Special thanks are also due to Donald R. Asmus, Scott Burke, Bhushan Gupta and his students at the Oregon Institute of Technology, and Carla Randall and her AP Calculus students.

Edition History

Edition 1 ..... July 1993
Edition 2 ..... January 1994
Edition 3 ..... May 1994
Edition 4 ..... December 1994

Contents

1. Programming

Understanding Programming 1-1
The Contents of a Program 1-1
Calculations in a Program 1-3
Entering and Executing Programs 1-4
Viewing and Editing Programs 1-9
Creating Programs on a Computer 1-10
Using Local Variables 1-11
Creating Local Variables 1-11
Evaluating Local Names 1-13
Defining the Scope of Local Variables 1-14
Compiled Local Variables 1-15
Creating User-Defined Functions as Programs . . . 1-16
Using Tests and Conditional Structures 1-17
Testing Conditions 1-17
Using Comparison Functions 1-17
Using Logical Functions 1-19
Testing Object Types 1-20
Testing Linear Structure 1-20
Using Conditional Structures and Commands 1-20
The IF ... THEN ... END Structure 1-20
The IFT Command 1-21
The IF ... THEN ... ELSE ... END Structure . 1-21
The IFTE Function 1-22
The CASE ... END Structure 1-22
Conditional Examples 1-23
Using Loop Structures 1-27
Using Definite Loop Structures 1-28
The START ... NEXT Structure 1-28
The START ... STEP Structure 1-30
The FOR ... NEXT Structure 1-32
The FOR ... STEP Structure 1-34

Using Indefinite Loop Structures 1-36
The DO ... UNTIL ... END Structure 1-36
The WHILE ... REPEAT ... END Structure . . 1-38
Using Loop Counters 1-39
Using Summations Instead of Loops 1-40
Using Flags 1-42
Types of Flags 1-42
Setting, Clearing, and Testing Flags 1-42
Recalling and Storing the Flag States 1-44
Using Subroutines 1-45
Single-Stepping through a Program 1-47
Trapping Errors 1-50
Causing and Analyzing Errors 1-51
Making an Error Trap 1-53
The IFERR ... THEN ... END Structure 1-53
The IFERR ... THEN ... ELSE ... END
Structure 1-54
Input 1-55
Data Input Commands 1-56
Using PROMPT ... CONT for Input 1-56
Using DISP FREEZE HALT ... CONT for Input . . 1-58
Using INPUT ... ENTER for Input 1-60
Using INFORM and CHOOSE for Input 1-67
Beeping to Get Attention 1-71
Stopping a Program for Keystroke Input 1-72
Using WAIT for Keystroke Input 1-72
Using KEY for Keystroke Input 1-73
Output 1-74
Data Output Commands 1-74
Labeling Output with Tags 1-74
Labeling and Displaying Output as Strings 1-75
Pausing to Display Output 1-76
Using MSGBOX to Display Output 1-77
Using Menus with Programs 1-77
Using Menus for Input 1-79
Using Menus to Run Programs 1-79
Turning Off the HP 48 from a Program 1-82

2. Programming Examples

Fibonacci Numbers 2-2
FIB1 (Fibonacci Numbers, Recursive Version) . . . 2-2
FIB2 (Fibonacci Numbers, Loop Version) 2-3
FIBT (Comparing Program-Execution Time) 2-5
Displaying a Binary Integer 2-7
PAD (Pad with Leading Spaces) 2-7
PRESERVE (Save and Restore Previous Status) . . 2-8
BDISP (Binary Display) 2-10
Median of Statistics Data 2-14
%TILE (Percentile of a List) 2-14
MEDIAN (Median of Statistics Data) 2-16
Expanding and Collecting Completely 2-19
MULTI (Multiple Execution) 2-19
EXCO (Expand and Collect Completely) 2-20
Minimum and Maximum Array Elements 2-22
MNX (Minimum or Maximum Element—Version 1) . 2-22
MNX2 (Minimum or Maximum Element—Version 2) 2-25
Applying a Program to an Array 2-29
Converting Between Number Bases 2-32
Verifying Program Arguments 2-36
NAMES (Check List for Exactly Two Names) . . . 2-36
VFY (Verify Program Argument) 2-38
Converting Procedures from Algebraic to RPN 2-40
Bessel Functions 2-43
Animation of Successive Taylor's Polynomials 2-45
SINTP (Converting a Plot to a Graphics Object) . . 2-45
SETTS (Superimposing Taylor's Polynomials) . . . 2-46
TSA (Animating Taylor's Polynomials) 2-47
Programmatic Use of Statistics and Plotting 2-49
Trace Mode 2-53
Inverse-Function Solver 2-54
Animating a Graphical Image 2-56

3. Command Reference

ABS 3-5
ACK 3-6
ACKALL 3-6
ACOS 3-7
ACOSH 3-9
ADD 3-11

ALOG 3-12
AMORT 3-12
AND 3-13
ANIMATE 3-14
APPLY 3-15
ARC 3-17
ARCHIVE 3-18
ARG 3-19
ARRY→ 3-19
→ARRY 3-20
ASIN 3-21
ASINH 3-23
ASN 3-24
ASR 3-25
ATAN 3-26
ATANH 3-28
ATICK 3-30
ATTACH 3-31
AUTO 3-32
AXES 3-33
BAR 3-34
BARPLOT 3-36
BAUD 3-36
BEEP 3-37
BESTFIT 3-37
BIN 3-38
BINS 3-38
BLANK 3-39
BOX 3-40
BUFLEN 3-40
BYTES 3-41
B→R 3-42
CASE 3-42
CEIL 3-44
CENTR 3-44
CF 3-45
CHOOSE 3-46
%CH 3-47
CHR 3-48
CKSM 3-49
CLEAR 3-50

CLKADJ 3-50
CLLCD 3-51
CLOSEIO 3-51
CLΣ 3-52
CLTEACH 3-52
CLUSR 3-52
CLVAR 3-53
CNRM 3-53
→COL 3-54
COL+ 3-54
COL− 3-55
COL→ 3-56
COLCT 3-56
COLΣ 3-57
COMB 3-58
CON 3-59
COND 3-60
CONIC 3-61
CONJ 3-62
CONLIB 3-63
CONST 3-63
CONT 3-64
CONVERT 3-65
CORR 3-65
COS 3-66
COSH 3-67
COV 3-67
CR 3-68
CRDIR 3-69
CROSS 3-69
CSWP 3-70
CYLIN 3-70
C→PX 3-71
C→R 3-71
DARCY 3-72
DATE 3-73
→DATE 3-73
DATE+ 3-74
DBUG 3-74
DDAYS 3-75
DEC 3-75

DECR 3-76
DEFINE 3-77
DEG 3-78
DELALARM 3-78
DELAY 3-79
DELKEYS 3-80
DEPND 3-81
DEPTH 3-82
DET 3-82
DETACH 3-84
DIAG→ 3-84
→DIAG 3-85
DIFFEQ 3-86
DISP 3-88
DO 3-89
DOERR 3-90
DOLIST 3-91
DOSUBS 3-92
DOT 3-94
DRAW 3-94
DRAX 3-95
DROP 3-95
DROPN 3-96
DROP2 3-96
DTAG 3-97
DUP 3-97
DUPN 3-98
DUP2 3-98
D→R 3-99
e 3-99
EGV 3-100
EGVL 3-101
ELSE 3-101
END 3-102
ENDSUB 3-102
ENG 3-103
EQ→ 3-104
EQNLIB 3-104
ERASE 3-105
ERRM 3-105
ERRN 3-106

ERR0 3-106
EVAL 3-107
EXP 3-108
EXPAN 3-109
EXPFIT 3-110
EXPM 3-110
EYEPT 3-111
F0λ 3-112
FACT 3-112
FANNING 3-113
FC? 3-114
FC?C 3-114
FFT 3-115
FINDALARM 3-116
FINISH 3-117
FIX 3-117
FLOOR 3-118
FOR 3-119
FP 3-121
FREE 3-121
FREE1 3-122
FREEZE 3-123
FS? 3-124
FS?C 3-125
FUNCTION 3-125
GET 3-127
GETI 3-129
GOR 3-130
GRAD 3-131
GRAPH 3-131
GRIDMAP 3-132
→GROB 3-133
GXOR 3-134
*H 3-135
HALT 3-136
HEAD 3-136
HEX 3-137
HISTOGRAM 3-137
HISTPLOT 3-139
HMS+ 3-139
HMS− 3-140

HMS→ 3-141
→HMS 3-142
HOME 3-142
i 3-143
IDN 3-143
IF 3-144
IFERR 3-146
IFFT 3-147
IFT 3-148
IFTE 3-149
IM 3-150
INCR 3-150
INDEP 3-151
INFORM 3-152
INPUT 3-154
INV 3-156
IP 3-157
IR 3-157
ISOL 3-158
KERRM 3-159
KEY 3-159
KGET 3-160
KILL 3-161
LABEL 3-162
LAST 3-162
LASTARG 3-163
LCD→ 3-163
→LCD 3-164
LIBEVAL 3-165
LIBS 3-165
LINE 3-166
ΣLINE 3-166
LINFIT 3-167
LININ 3-168
LIST→ 3-169
→LIST 3-169
ΣLIST 3-170
ΔLIST 3-170
ΠLIST 3-171
LN 3-172
LNP1 3-174

LOG 3-174
LOGFIT 3-176
LQ 3-176
LR 3-177
LSQ 3-178
LU 3-179
MANT 3-179
↑MATCH 3-180
↓MATCH 3-181
MAX 3-183
MAXR 3-183
MAXΣ 3-184
MCALC 3-185
MEAN 3-185
MEM 3-186
MENU 3-187
MERGE 3-190
MERGE1 3-190
MIN 3-191
MINEHUNT 3-192
MINIT 3-193
MINR 3-193
MINΣ 3-194
MITM 3-194
MOD 3-195
MROOT 3-195
MSGBOX 3-196
MSOLVR 3-197
MUSER 3-197
NDIST 3-198
NEG 3-199
NEWOB 3-200
NEXT 3-201
NEXT 3-201
NOT 3-201
NOVAL 3-203
NSUB 3-203
NUM 3-204
→NUM 3-207
NUMX 3-207
NUMY 3-208

NΣ 3-208
OBJ→ 3-209
OCT 3-210
OFF 3-211
OLDPRT 3-211
OPENIO 3-212
OR 3-213
ORDER 3-214
OVER 3-215
PARAMETRIC 3-215
PARITY 3-217
PARSURFACE 3-218
PATH 3-219
PCOEF 3-220
PCONTOUR 3-221
PCOV 3-222
PDIM 3-223
PERM 3-224
PEVAL 3-224
PGDIR 3-225
PICK 3-226
PICT 3-226
PICTURE 3-227
PINIT 3-228
PIXOFF 3-228
PIXON 3-229
PIX? 3-229
PKT 3-230
PMAX 3-231
PMIN 3-231
POLAR 3-232
POS 3-234
PREDV 3-234
PREDX 3-235
PREDY 3-236
PRLCD 3-237
PROMPT 3-238
PROOT 3-238
PRST 3-239
PRSTC 3-240
PRVAR 3-240

PR1 3-241
PSDEV 3-242
PURGE 3-243
PUT 3-244
PUTI 3-246
PVAR 3-247
PVARS 3-248
PVIEW 3-249
PWRFIT 3-250
PX→C 3-250
→Q 3-251
→Qπ 3-251
QR 3-253
QUAD 3-253
QUOTE 3-254
RAD 3-256
RAND 3-256
RANK 3-257
RANM 3-257
RATIO 3-258
RCEQ 3-259
RCI 3-260
RCIJ 3-260
RCL 3-261
RCLALARM 3-262
RCLF 3-262
RCLKEYS 3-263
RCLMENU 3-264
RCLΣ 3-264
RCWS 3-265
RDM 3-266
RDZ 3-267
RE 3-267
RECN 3-268
RECT 3-269
RECV 3-269
REPEAT 3-270
REPL 3-270
RES 3-271
RESTORE 3-273
REVLIST 3-274

RKF 3-274
RKFERR 3-276
RKFSTEP 3-277
RL 3-278
RLB 3-278
RND 3-279
RNRM 3-280
ROLL 3-280
ROLLD 3-281
ROOT 3-281
ROT 3-282
→ROW 3-282
ROW+ 3-283
ROW− 3-284
ROW→ 3-284
RR 3-285
RRB 3-285
RREF 3-286
RRK 3-286
RRKSTEP 3-288
RSBERR 3-290
RSD 3-291
RSWP 3-292
R→B 3-292
R→C 3-293
R→D 3-294
SAME 3-294
SBRK 3-295
SCALE 3-295
SCATRPLOT 3-296
SCATTER 3-297
SCHUR 3-298
SCI 3-299
SCLΣ 3-299
SCONJ 3-300
SDEV 3-300
SEND 3-301
SEQ 3-302
SERVER 3-303
SF 3-304
SHOW 3-305

SIDENS 3-305
SIGN 3-306
SIMU 3-307
SIN 3-307
SINH 3-308
SINV 3-309
SIZE 3-309
SL 3-310
SLB 3-311
SLOPEFIELD 3-312
SNEG 3-313
SNRM 3-314
SOLVEQN 3-314
SORT 3-315
SPHERE 3-316
SQ 3-316
SR 3-317
SRAD 3-317
SRB 3-318
SRECV 3-318
SST 3-320
SST↓ 3-320
START 3-321
STD 3-322
STEP 3-323
STEQ 3-324
STIME 3-324
STO 3-325
STOALARM 3-326
STOF 3-327
STOKEYS 3-328
STO+ 3-329
STO− 3-330
STO∗ 3-330
STO/ 3-331
STOΣ 3-332
STREAM 3-333
STR→ 3-333
→STR 3-334
STWS 3-335
SUB 3-336

SVD 3-337
SVL 3-338
SWAP 3-338
SYSEVAL 3-339
%T 3-339
→TAG 3-341
TAIL 3-341
TAN 3-342
TANH 3-343
TAYLR 3-343
TDELTA 3-344
TEACH 3-345
TEXT 3-345
THEN 3-346
TICKS 3-346
TIME 3-347
→TIME 3-347
TINC 3-348
TLINE 3-349
TMENU 3-350
TOT 3-351
TRACE 3-351
TRANSIO 3-352
TRN 3-353
TRNC 3-353
TRUTH 3-355
TSTR 3-356
TVARS 3-357
TVM 3-357
TVMBEG 3-358
TVMEND 3-358
TVMROOT 3-358
TYPE 3-359
UBASE 3-360
UFACT 3-361
→UNIT 3-362
UNTIL 3-362
UPDIR 3-363
UTPC 3-363
UTPF 3-364
UTPN 3-365

UTPT 3-365
UVAL 3-366
VAR 3-367
VARS 3-367
VERSION 3-368
VTYPE 3-368
→V2 3-369
→V3 3-370
V→ 3-371
*W 3-372
WAIT 3-373
WHILE 3-374
WIREFRAME 3-375
WSLOG 3-377
ΣX 3-379
ΣX^2 3-379
XCOL 3-380
XMIT 3-381
XOR 3-382
XPON 3-383
XRECV 3-384
XRNG 3-384
XROOT 3-385
XSEND 3-386
XVOL 3-386
XXRNG 3-387
ΣX*Y 3-388
ΣY 3-388
ΣY^2 3-389
YCOL 3-389
YRNG 3-390
YSLICE 3-391
YVOL 3-392
YYRNG 3-393
ZFACTOR 3-393
ZVOL 3-394
+ 3-395
− 3-397
* 3-399
/ 3-401
^ 3-402

< 3-403
≤ 3-404
> 3-406
≥ 3-407
= 3-408
== 3-410
≠ 3-411
! 3-412
∫ 3-413
∂ 3-415
% 3-416
π 3-417
Σ 3-418
Σ+ 3-419
Σ− 3-420
√ 3-420
_ 3-423
| (Where) 3-423
→ 3-424

4. Equation Reference

Columns and Beams (1) 4-1
Elastic Buckling (1, 1) 4-3
Eccentric Columns (1, 2) 4-3
Simple Deflection (1, 3) 4-4
Simple Slope (1, 4) 4-5
Simple Moment (1, 5) 4-6
Simple Shear (1, 6) 4-6
Cantilever Deflection (1, 7) 4-7
Cantilever Slope (1, 8) 4-7
Cantilever Moment (1, 9) 4-8
Cantilever Shear (1, 10) 4-9
Electricity (2) 4-9
Coulomb's Law (2, 1) 4-11
Ohm's Law and Power (2, 2) 4-11
Voltage Divider (2, 3) 4-12
Current Divider (2, 4) 4-12
Wire Resistance (2, 5) 4-13
Series and Parallel R (2, 6) 4-13
Series and Parallel C (2, 7) 4-14
Series and Parallel L (2, 8) 4-14

Capacitive Energy (2, 9) 4-15
Inductive Energy (2, 10) 4-15
RLC Current Delay (2, 11) 4-16
DC Capacitor Current (2, 12) 4-16
Capacitor Charge (2, 13) 4-17
DC Inductor Voltage (2, 14) 4-17
RC Transient (2, 15) 4-18
RL Transient (2, 16) 4-18
Resonant Frequency (2, 17) 4-19
Plate Capacitor (2, 18) 4-19
Cylindrical Capacitor (2, 19) 4-20
Solenoid Inductance (2, 20) 4-20
Toroid Inductance (2, 21) 4-21
Sinusoidal Voltage (2, 22) 4-21
Sinusoidal Current (2, 23) 4-21
Fluids (3) 4-22
Pressure at Depth (3, 1) 4-23
Bernoulli Equation (3, 2) 4-23
Flow with Losses (3, 3) 4-24
Flow in Full Pipes (3, 4) 4-26
Forces and Energy (4) 4-27
Linear Mechanics (4, 1) 4-28
Angular Mechanics (4, 2) 4-29
Centripetal Force (4, 3) 4-29
Hooke's Law (4, 4) 4-30
1D Elastic Collisions (4, 5) 4-30
Drag Force (4, 6) 4-31
Law of Gravitation (4, 7) 4-31
Mass-Energy Relation (4, 8) 4-31
Gases (5) 4-32
Ideal Gas Law (5, 1) 4-33
Ideal Gas State Change (5, 2) 4-33
Isothermal Expansion (5, 3) 4-34
Polytropic Processes (5, 4) 4-34
Isentropic Flow (5, 5) 4-35
Real Gas Law (5, 6) 4-36
Real Gas State Change (5, 7) 4-36
Kinetic Theory (5, 8) 4-37
Heat Transfer (6) 4-37
Heat Capacity (6, 1) 4-38
Thermal Expansion (6, 2) 4-39

Conduction (6, 3) 4-39
Convection (6, 4) 4-40
Conduction + Convection (6, 5) 4-41
Black Body Radiation (6, 6) 4-42
Magnetism (7) 4-43
Straight Wire (7, 1) 4-43
Force between Wires (7, 2) 4-44
Magnetic (B) Field in Solenoid (7, 3) 4-44
Magnetic (B) Field in Toroid (7, 4) 4-45
Motion (8) 4-46
Linear Motion (8, 1) 4-47
Object in Free Fall (8, 2) 4-47
Projectile Motion (8, 3) 4-48
Angular Motion (8, 4) 4-48
Circular Motion (8, 5) 4-49
Terminal Velocity (8, 6) 4-49
Escape Velocity (8, 7) 4-49
Optics (9) 4-50
Law of Refraction (9, 1) 4-51
Critical Angle (9, 2) 4-51
Brewster's Law (9, 3) 4-52
Spherical Reflection (9, 4) 4-52
Spherical Refraction (9, 5) 4-53
Thin Lens (9, 6) 4-54
Oscillations (10) 4-54
Mass-Spring System (10, 1) 4-55
Simple Pendulum (10, 2) 4-56
Conical Pendulum (10, 3) 4-56
Torsional Pendulum (10, 4) 4-57
Simple Harmonic (10, 5) 4-57
Plane Geometry (11) 4-58
Circle (11, 1) 4-59
Ellipse (11, 2) 4-59
Rectangle (11, 3) 4-60
Regular Polygon (11, 4) 4-61
Circular Ring (11, 5) 4-61
Triangle (11, 6) 4-62
Solid Geometry (12) 4-63
Cone (12, 1) 4-64
Cylinder (12, 2) 4-64
Parallelepiped (12, 3) 4-65

Sphere (12, 4) 4-66
Solid State Devices (13) 4-67
PN Step Junctions (13, 1) 4-69
NMOS Transistors (13, 2) 4-71
Bipolar Transistors (13, 3) 4-73
JFETs (13, 4) 4-74
Stress Analysis (14) 4-76
Normal Stress (14, 1) 4-77
Shear Stress (14, 2) 4-77
Stress on an Element (14, 3) 4-78
Mohr's Circle (14, 4) 4-79
Waves (15) 4-80
Transverse Waves (15, 1) 4-80
Longitudinal Waves (15, 2) 4-81
Sound Waves (15, 3) 4-81
References 4-82

A. Error and Status Messages

B. Table of Units

C. System Flags

D. Reserved Variables

Contents of the Reserved Variables D-2
ALRMDAT D-2
CST D-3
"der-" Names D-4
EQ D-4
EXPR D-5
IOPAR D-5
MHpar D-7
Mpar D-7
n1, n2, ... D-7
Nmines D-8
PPAR D-8
PRTPAR D-11
s1, s2, ... D-12
VPAR D-13
ZPAR D-14
ΣDAT D-15
ΣPAR D-16

E. New Commands

F. Technical Reference

Object Sizes F-2
Automatic Simplification Rules F-3
Symbolic Integration Patterns F-5
Trigonometric Expansions F-7
Source References F-9

G. Parallel Processing with Lists

Index

1

Programming

If you've used a calculator or computer before, you're probably familiar with the idea of *programs*. Generally speaking, a program is something that gets the calculator or computer to do certain tasks for you—more than a built-in command might do. In the HP 48, a program is an *object* that does the same thing.

Understanding Programming

An HP 48 program is an object with « » delimiters containing a sequence of numbers, commands, and other objects you want to execute automatically to perform a task.

For example, a program that takes a number from that stack, finds its factorial, and divides the result by 2 would look like this: « ! 2 / »
or

```
«
  !
  2
  /
»
```

The Contents of a Program

As mentioned above, a program contains a sequence of objects. As each object is processed in a program, the action depends on the type of object, as summarized below.

Actions for Certain Objects in Programs

Object	Action
Command	*Executed.*
Number	Put on the stack.
Algebraic	Put on the stack.
String	Put on the stack.
List	Put on the stack.
Program	Put on the stack.
Global name (quoted)	Put on the stack.
Global name (unquoted)	■ Program *executed.* ■ Name evaluated. ■ Directory becomes current. ■ Other object put on the stack.
Local name (quoted)	Put on the stack.
Local name (unquoted)	Contents put on the stack.

As you can see from this table, most types of objects are simply put on the stack—but built-in commands and programs called by name cause *execution*. The following examples show the results of executing programs containing different sequences of objects.

Examples of Program Actions

Program	Results
« 1 2 »	2: 1 1: 2
« "Hello" { A B } »	2: "Hello" 1: { A B }
« '1+2' »	1: '1+2'
« '1+2' →NUM »	1: 3
« « 1 2 + » »	1: « 1 2 + »
« « 1 2 + » EVAL »	1: 3

1-2 Programming

Programs can also contain *structures*. A structure is a program segment with a defined organization. Two basic kinds of structures are available:

- **Local variable structure.** The ↠ command defines local variable names and a corresponding algebraic or program object that's evaluated using those variables.
- **Branching structures.** Structure words (like DO ... UNTIL ... END) define conditional or loop structures to control the order of execution within a program.

A *local variable structure* has one of the following organizations inside a program:

« ↠ $name_1$ ... $name_n$ *'algebraic'* »
« ↠ $name_1$ ... $name_n$ « *program* » »

The → command removes n objects from the stack and stores them in the named local variables. The algebraic or program object in the structure is *automatically evaluated* because it's an element of the structure—even though algebraic and program objects are put on the stack in other situations. Each time a local variable name appears in the algebraic or program object, the variable's contents are substituted.

So the following program takes two numbers from the stack and returns a numeric result:

```
« → a b 'ABS(a-b)' »
```

Calculations in a Program

Many calculations in programs take data from the stack. Two typical ways to manipulate stack data are:

- **Stack commands.** Operate directly on the objects on the stack.
- **Local variable structures.** Stores the stack objects in temporary local variables, then uses the variable names to represent the data in the following algebraic or program object.

Numeric calculations provide convenient examples of these methods. The following programs use two numbers from the stack to calculate the hypotenuse of a right triangle using the formula $\sqrt{x^2 + y^2}$.

```
« SQ SWAP SQ + √ »
« → x y « x SQ y SQ + √ » »
« → x y '√(x^2+y^2)' »
```

The first program uses stack commands to manipulate the numbers on the stack—the calculation uses stack syntax. The second program uses a local variable structure—the calculation uses algebraic syntax. Note that the underlying formula is most apparent in the third program. This third method is often the easiest to write, read, and debug.

Entering and Executing Programs

A program is an object—it occupies one level on the stack, and you can store it in a variable.

To enter a program:

1. Press ◀ « ». The PRG annunciator appears, indicating Program-entry mode is active.
2. Enter the commands and other objects (with appropriate delimiters) in order for the operations you want the program to execute.
   - Press SPC to separate consecutive numbers.
   - Press ▶ to move past closing delimiters.
3. Optional: Press ▶ ↵ (newline) to start a new line in the command line at any time.
4. Press ENTER to put the program on the stack.

In Program-entry mode (PRG annunciator on), command keys aren't executed—they're entered in the command line instead. Only nonprogrammable operations such as ◀ and VAR are executed.

Line breaks are discarded when you press ENTER.

To enter commands and other objects in a program:

- Press the keyboard or menu key for the command or object.

  or
- Type the characters using the alpha keyboard.

To store or name a program:

1. Enter the program on the stack.
2. Enter the variable name (with ' delimiters) and press [STO].

You can choose descriptive names for programs. Here are some ideas of what the name can describe:

- The calculation or action. Examples: *SPH* (spherical-cap volume), *SORT* (sort a list).
- The input and output. Examples: $X \rightarrow FX$ (x to $f(x)$), $RH \rightarrow V$ (radius-and-height to volume).
- The technique. Example: *SPHLV* (spherical-cap volume using local variables).

To execute a program:

- Press [VAR] then the menu key for the program name.
  or
- Enter the program name (with *no* delimiters) and press [ENTER].
  or
- Put the program name in level 1 and press [EVAL].
  or
- Put the program object in level 1 and press [EVAL].

To stop an executing program:

- Press [CANCEL].

Example: Enter a program that takes a radius value from the stack and calculates the volume of a sphere of radius r using

$$V = \frac{4}{3}\pi r^3$$

If you were going to calculate the volume manually after entering the radius on the stack, you might press these keys:

3 [y^x] [◀] [π] [×] 4 [ENTER] 3 [÷] [×] [◀] [→NUM]

Enter the same keystrokes in a program. (↷ ↵ just starts a new line.)

◀ « »
3 y^x ◀ π ✕ 4 SPC 3 ÷ ✕
↷ ↵ ◀ →NUM



Put the program on the stack.

ENTER



Store the program in variable *VOL*. Then put a radius of 4 on the stack and run the *VOL* program.

' VOL STO
4 VAR VOL



The program is

« 3 ^ π * 4 3 / * →NUM »

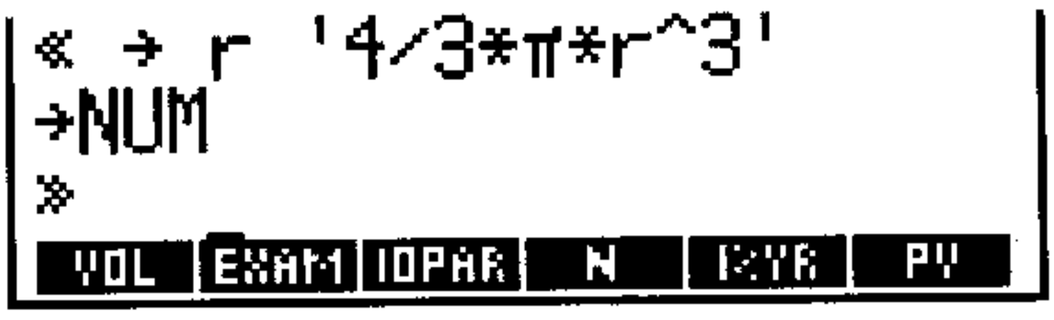
Example: Replace the program from the previous example with one that's easier to read. Enter a program that uses a local variable structure to calculate the volume of a sphere. The program is

« → r '4/3*π*r^3' →NUM »

(You need to include →NUM because π causes a symbolic result.)

Enter the program. (↷ ↵ just starts a new line.)

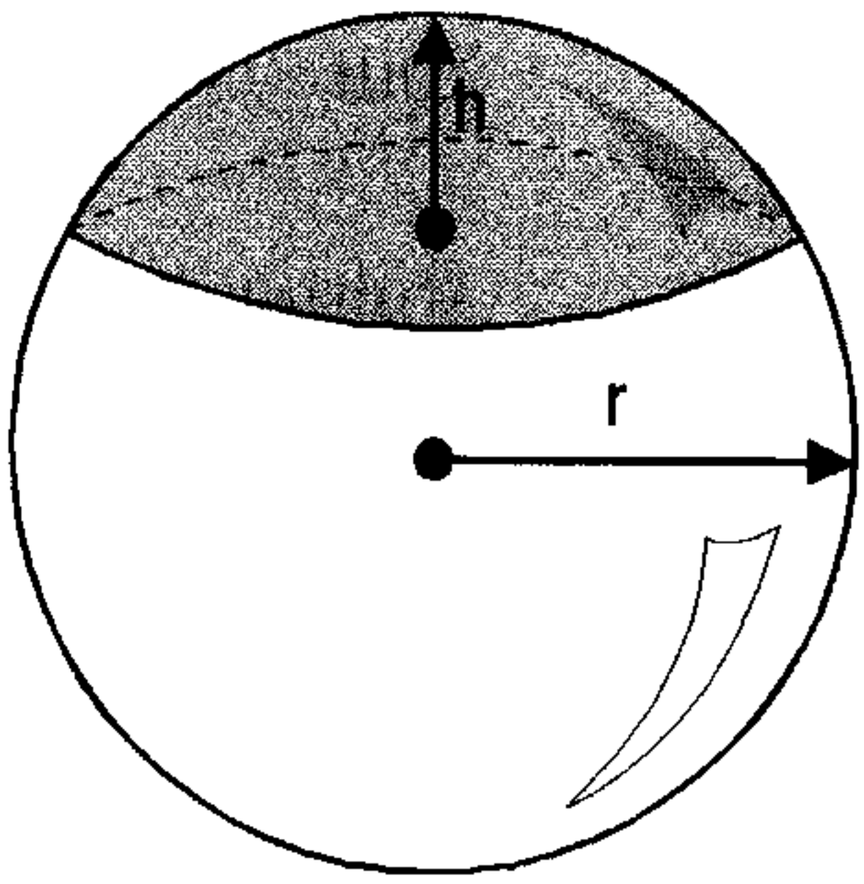
◀ « »
↷ → r SPC
' 4 ÷ 3 ✕ ◀ π ✕
r y^x 3 ▶ ↷ ↵
◀ →NUM



Put the program on the stack, store it in *VOL*, and calculate the volume for a radius of 4.

4 VOL

Example: Enter a program *SPH* that calculates the volume of a spherical cap of height h within a sphere of radius R using values stored in variables H and R.



$$V = \frac{1}{3}\pi h^2(3r - h)$$

In this and following chapters on programming, "stack diagrams" show what arguments must be on the stack before a program is executed and what results the program leaves on the stack. Here's the stack diagram for *SPH*.

Level 1	→	Level 1
	→	*volume*

The diagram indicates that *SPH* takes no arguments from the stack and returns the volume of the spherical cap to level 1. (*SPH* assumes that you've stored the numerical value for the radius in variable R and the numerical value for the height in variable H. These are *global* variables—they exist outside the program.)

Program listings are shown with program steps in the left column and associated comments in the right column. Remember, you can either press the command keys or type in the command names to key in the program. In this first listing, the keystrokes are also shown.

Program:	Keys:	Comments:
«	◀ « »	Begins the program.
'1/3	' 1 ÷ 3	Begins the algebraic expression to calculate the volume.
*π*H^2	× ◀ π × H y^x 2	Multiplies by πh^2.
*(3*R−H)'	× ◀ ( ) 3 × R − H ▶ ▶	Multiplies by $3r - h$, completing the calculation and ending the expression.
→NUM	◀ →NUM	Converts the expression with π to a number.
»		Ends the program.
	ENTER	Puts the program on the stack.
	' SPH STO	Stores the program in variable *SPH*.

This is the program:

```
« '1/3*π*H^2*(3*R−H)' →NUM »
```

Now use *SPH* to calculate the volume of a spherical cap of radius $r = 10$ and height $h = 3$.

First, store the data in the appropriate variables. Then select the VAR menu and execute the program. The answer is returned to level 1 of the stack.

10 ' R STO
3 ' H STO
VAR SPH

```
1:                254.469004942
 H   R   SPH  VOL  EXAM  LOPAR
```

1-8 Programming

Viewing and Editing Programs

You view and edit programs the same way you view and edit other objects—using the command line.

To view or edit a program:

1. View the program:
   - If the program is in level 1, press ◀ EDIT, or ▼.
   - If the program is stored in a variable, use the Memory Browser (▶ MEMORY) to view the variable, or press VAR ▶ and the variable's menu key, followed by ▼.
2. Optional: Make changes.
3. Press ENTER to save any changes (or press CANCEL to discard changes) and return to the stack.

The Memory Browser lets you change a stored program without having to do a store operation. ◀ EDIT lets you change a program and then store the new version in a different variable.

While you're editing a program, you may want to switch the command-line entry mode between Program-entry mode (for editing most objects) and Algebraic/Program-entry mode (for editing algebraic objects). The PRG and ALG annunciators indicate the current mode.

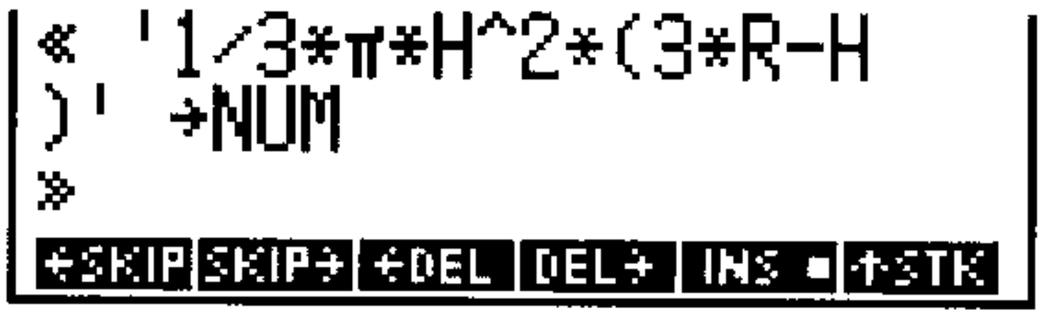
To switch between entry modes:

- Press ▶ ENTRY.

Example: Edit *SPH* from the previous example so that it stores the number from level 1 into variable *H* and the number from level 2 into variable *R*.

Use EDIT to start editing *SPH*.

VAR
▶ SPH ▼



Move the cursor past the first program delimiter and insert the new program steps.

▶ ' H ▶ STO
' R ▶ STO



Save the edited version of *SPH* in the variable. Then, to verify that the changes were saved, view *SPH* in the command line.

ENTER VAR ↶ SPH
↷ SPH ▼



Press CANCEL to stop viewing.

Creating Programs on a Computer

It is convenient to create programs and other objects on a computer and then load them into the HP 48 using the calculator's serial port.

If you are creating programs on a computer, you can include "comments" in the computer version of the program.

To include a comment in a program:

- Enclose the comment text between two @ characters.

  or

- Enclose the comment text between one @ character and the end of the line.

Whenever the HP 48 processes text entered in the command line—either from keyboard entry or transferred from a computer—it strips away the @ characters and the text they surround. However, @ characters are not affected if they're inside a string.

Using Local Variables

The program *SPH* in the previous example uses global variables for data storage and recall. There are disadvantages to using global variables in programs:

- After program execution, global variables that you no longer need to use must be purged if you want to clear the VAR menu and free user memory.
- You must explicitly store data in global variables prior to program execution, or have the program execute STO.

Local variables address the disadvantages of global variables in programs. Local variables are temporary variables *created by a program*. They exist only while the program is being executed and cannot be used outside the program. They never appear in the VAR menu. In addition, local variables are accessed faster than global variables. (By convention, this manual uses lowercase names for local variables.) A compiled local variable is a form of local variable that can be used outside of the program that creates it. See "Compiled Local Variables" on page 1-15 for more information.

Creating Local Variables

In a program, a *local variable structure* creates local variables.

To enter a local variable structure in a program:

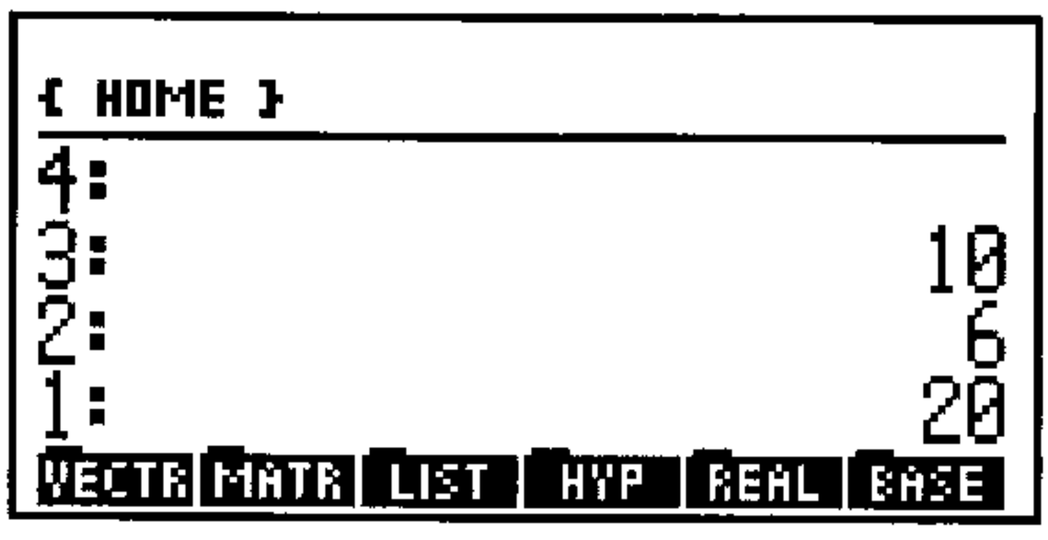
1. Enter the → command (press).
2. Enter one or more variable names.
3. Enter a *defining procedure* (an algebraic or program object) that uses the names.

   « → $name_1$ $name_2$ ... $name_n$ '*algebraic*' »

   or

   « → $name_1$ $name_2$ ... $name_n$ « *program* » »

When the → command is executed in a program, n values are taken from the stack and assigned to variables $name_1$, $name_2$, ... $name_n$. For example, if the stack looks like this:



then

- → a creates local variable $a = 20$.
- → a b creates local variables $a = 6$ and $b = 20$.
- → a b c creates local variables $a = 10$, $b = 6$, and $c = 20$.

The defining procedure then uses the local variables to do calculations.

Local variable structures have these advantages:

- The → command stores the values from the stack in the corresponding variables—you don't need to explicitly execute STO.
- Local variables automatically disappear when the defining procedure for which they are created has completed execution. Consequently, local variables don't appear in the VAR menu, and they occupy user memory only during program execution.
- Local variables exist only within their defining procedure—different local variable structures can use the same variable names without conflict.

Example: The following program *SPHLV* calculates the volume of a spherical cap using local variables. The defining procedure is an algebraic expression.

Level 2	Level 1	→	Level 1
r	*h*	→	*volume*

Program:	**Comments:**
«	
→ r h	Creates local variables r and h for the radius of the sphere and height of the cap.
'1/3*π*h^2*(3*r-h)'	Expresses the defining procedure. In this program, the defining procedure for the local variable structure is an algebraic expression.
→NUM	Converts expression to a number.
»	
ENTER ' SPHLV STO	Stores the program in variable *SPHLV*.

Now use *SPHLV* to calculate the volume of a spherical cap of radius $r = 10$ and height $h = 3$. Enter the data on the stack in the correct order, then execute the program.

10 ENTER 3
VAR SPHLV



Evaluating Local Names

Local names are evaluated differently from global names. When a global name is evaluated, the object stored in the corresponding variable is itself evaluated. (You've seen how programs stored in global variables are automatically evaluated when the name is evaluated.)

When a local name is evaluated, the object stored in the corresponding variable is returned to the stack but is *not* evaluated. When a local variable contains a number, the effect is identical to evaluation of a global name, since putting a number on the stack is equivalent to evaluating it. However, if a local variable contains a program, algebraic expression, or global variable name—and if you want it evaluated—the program should execute EVAL after the object is put on the stack.

Defining the Scope of Local Variables

Local variables exist *only* inside the defining procedure.

Example: The following program excerpt illustrates the availability of local variables in *nested* defining procedures (procedures within procedures). Because local variables a, b, and c already exist when the defining procedure for local variables d, e, and f is executed, they're available for use in that procedure.

Program:	**Comments:**
※	
:	
→ a b c	No local variables are available.
※	Defines local variables a, b, c.
a b + c +	Local variables a, b, c are available in this procedure.
→ d e f	Defines local variables d, e, f.
'a/(d*e+f)'	Local variables a, b, c and d, e, f are available in this procedure.
a c / −	Only local variables a, b, c are available.
※	
:	No local variables are available.
※	

Example: In the following program excerpt, the defining procedure for local variables d, e, and f calls a program that you previously created and stored in global variable *P1*.

Program:

```
«
  :
  → a b c
  «
    a b + c +
    → d e f
    'P1+a/(d*e+f)'

    a c / -
  »
  :
»
```

Comments:

Defines local variables *d*, *e*, *f*.

Local variables *a*, *b*, *c* and *d*, *e*, *f* are available in this procedure. The defining procedure executes the program stored in variable *P1*.

The six local variables are *not* available in program *P1* because they didn't exist when you created *P1*. The objects stored in the local variables are available to program *P1* only if you put those objects on the stack for *P1* to use or store those objects in global variables.

Conversely, program *P1* can create its own local variable structure (with any names, such as *a*, *c*, and *f*, for example) without conflicting with the local variables of the same name in the procedure that calls *P1*. It is possible to create a special type of local variable that can be used in other programs or subroutines. This type of local variable is called a compiled local variable.

Compiled Local Variables

Global variables use up memory, and local variables can't be used outside of the program they were created in. Compiled local variables bridge the gap between these two variable types. To programs, compiled local variables look like global variables, but to the calculator they act like local variables. This means you can create a compiled local variable in a local variable structure, use it in any other program that is called within that structure, and when the program finishes, the variable is gone.

Compiled local variables have a special naming convention: they must begin with a ←. For example,

```
«
  → ←y
'IFTE(←y<0,BELOW,ABOVE)'
»
```

The variable ←y is a compiled local variable that can be used in the two programs BELOW and ABOVE.

Creating User-Defined Functions as Programs

The defining procedure for a local variable structure can be either an algebraic or program object.

A program that consists solely of a local variable structure whose defining procedure is an algebraic expression is a user-defined function.

If a program begins with a local variable structure and has a program as the defining procedure, the complete program acts like a user-defined function in two ways: it takes numeric or symbolic arguments, and takes those arguments either from the stack or in algebraic syntax. However, it does *not* have a derivative. (The defining program must, like algebraic defining procedures, return only *one* result to the stack.)

There's an advantage to using a program as the defining procedure for a local variable structure: The program can contain commands not allowed in algebraic expressions. For example, loop structures are not allowed in algebraic expressions.

Using Tests and Conditional Structures

You can use commands and branching structures that let programs ask questions and make decisions. *Comparison functions* and *logical functions* test whether or not specified conditions exist. *Conditional structures* and *conditional commands* use test results to make decisions.

Testing Conditions

A test is an algebraic or a command sequence that returns a *test result* to the stack. A test result is either *true*—indicated by a value of 1—or it is *false*—indicated by a value of 0.

To include a test in a program:

- To use stack syntax, enter the two arguments, then enter the test command.
- To use algebraic syntax, enter the test expression (with ' delimiters).

You often use test results in conditional structures to determine which clause of the structure to execute. Conditional structures are described under "Using Conditional Structures and Commands" on page 1-20.

Example: Test whether or not X is less than Y. To use stack syntax, enter `X Y <`. To use algebraic syntax, enter `'X<Y'`. (For both cases, if X contains 5 and Y contains 10, then the test is true and `1` is returned to the stack.)

Using Comparison Functions

Comparison functions compare two objects, using either stack syntax or algebraic syntax.

Comparison Functions

Key	Programmable Command	Description
PRG TEST (pages 1 and 2):		
==	==	Tests equality of two objects.
≠	≠	Not equal.
<	<	Less than.
>	>	Greater than.
≤	≤	Less than or equal to.
≥	≥	Greater than or equal to.
SAME	SAME	Identical. Like ==, but doesn't allow a comparison between the numerical value of an algebraic (or name) and a number. Also considers the wordsize of a binary integer.

The comparison commands return 1 (true) or 0 (false) based on the comparison—or an expression that can evaluate to 1 or 0. The order of the comparison is "level 2 *test* level 1," where *test* is the comparison function.

All comparison commands except SAME return the following:

- If neither object is an algebraic or a name, returns 1 if the two objects are the same type and have the same value, or 0 otherwise. For example, if 6 is stored in X, `X 5 <` puts 6 and 5 on the stack, then removes them and returns 0. (Lists and programs are considered to have the same value if the objects they contain are identical. For strings, "less than" means "alphabetically previous.")
- If one object is an algebraic (or name) and the other object is an algebraic (or name) or a number, returns an expression that must be evaluated to get a test result based on numeric values. For example, if 6 is stored in X, `'X' 5 <` returns `'X<5'`, then →NUM returns 0.

(Note that == is used for comparisons, while = separates two sides of an equation.)

SAME returns 1 (true) if two objects are identical. For example, `'X+3' 4 SAME` returns 0 regardless of the value of X because the algebraic `'X+3'` is not identical to the real number 4. Binary integers

must have the same wordsize and the same value to be identical. For all object types other than algebraics, names, and binary integers, SAME works just like ==.

You can use any comparison function (except SAME) in an algebraic by putting it *between* its two arguments. For example, if 6 is stored in X, 'X<5' →NUM returns 0.

Using Logical Functions

Logical functions return a test result based on the outcomes of two previously executed tests. Note that these four functions interpret *any nonzero argument* as a true result.

Logical Functions

Keys	Programmable Command	Description
PRG TEST (page 2):		
AND	AND	Returns 1 (true) only if both arguments are true.
OR	OR	Returns 1 (true) if either or both arguments are true.
XOR	XOR	Returns 1 (true) if either argument, but not both, is true.
NOT	NOT	Returns 1 (true) if the argument is 0 (false); otherwise, returns 0 (false).

AND, OR, and XOR combine two test results. For example, if 4 is stored in Y, Y 8 < 5 AND returns 1. First, Y 8 < returns 1 to the stack. AND removes 1 and 5 from the stack, interpreting both as true results, and returns 1 to the stack.

NOT returns the logical inverse of a test result. For example, if 1 is stored in X and 2 is stored in Y, X Y < NOT returns 0.

You can use AND, OR, and XOR in algebraics as *infix* functions. For example, '3<5 XOR 4>7' →NUM returns 1.

You can use NOT as a *prefix* function in algebraics. For example, 'NOT Z≤4' →NUM returns 0 if $Z = 2$.

Testing Object Types

The TYPE command (PRG TEST NXT TYPE) takes any object as its argument and returns the number that identifies that object type. For example, "HELLO" TYPE returns 2, the value for a string object. See the table of object types in chapter 3, in the TYPE command, to find HP 48 objects and their corresponding type numbers.

Testing Linear Structure

The LININ command (PRG NXT TEST ↷ PREV LININ) takes an algebraic equation on level 2 and an variable on level 1 as arguments and returns 1 if the equation is linear for that variable, or 0 if it is not. For example, 'H+Y^2' 'H' LININ returns 1 because the equation is structurally linear for H. See the LININ command in chapter 3 for more information.

Using Conditional Structures and Commands

Conditional structures let a program make a decision based on the results of tests.

Conditional commands let you execute a true-clause or a false-clause (each of which are a *single* command or object).

These conditional structures and commands are contained in the PRG BRCH menu (PRG BRCH):

- IF ... THEN ... END structure.
- IF ... THEN ... ELSE ... END structure.
- CASE ... END structure.
- IFT (if-then) command.
- IFTE (if-then-else) function.

The IF ... THEN ... END Structure

The syntax for this structure is

« ... IF *test-clause* THEN *true-clause* END ... »

IF ... THEN ... END executes the sequence of commands in the *true-clause* only if the *test-clause* evaluates to true. The test-clause can be a command sequence (for example, A B ≤) or an algebraic (for

example, 'A≤B'). If the test-clause is an algebraic, it's *automatically evaluated* to a number—you don't need →NUM or EVAL.

IF begins the test-clause, which leaves a test result on the stack. THEN removes the test result from the stack. If the value is nonzero, the true-clause is executed—otherwise, program execution resumes following END. See "Conditional Examples" on page 1-23.

To enter IF ... THEN ... END in a program:

- Press PRG BRCH ◀ IF.

The IFT Command

The IFT command takes two arguments: a *test-result* in level 2 and a *true-clause* object in level 1. If the test-result is true, the true-clause object is executed—otherwise, the two arguments are removed from the stack. See "Conditional Examples" on page 1-23.

To enter IFT in a program:

- Press PRG BRCH NEXT ◀ IFT.

The IF ... THEN ... ELSE ... END Structure

The syntax for this structure is

« ... IF *test-clause*
THEN *true-clause* ELSE *false-clause* END ... »

IF ... THEN ... ELSE ... END executes either the *true-clause* sequence of commands if the *test-clause* is true, or the *false-clause* sequence of commands if the *test-clause* is false. If the test-clause is an algebraic, it's automatically evaluated to a number—you don't need →NUM or EVAL.

IF begins the test-clause, which leaves a test result on the stack. THEN removes the test result from the stack. If the value is nonzero, the true-clause is executed—otherwise, the false-clause is executed. After the appropriate clause is executed, execution resumes following END. See "Conditional Examples" on page 1-23.

To enter IF ... THEN ... ELSE ... END in a program:

- Press PRG BRCH ▶ IF.

The IFTE Function

The algebraic syntax for this function is

'IFTE(*test*, *true-clause*, *false-clause*)'

If *test* evaluates true, the *true-clause* algebraic is evaluated—otherwise, the *false-clause* algebraic is evaluated.

You can also use the IFTE function with stack syntax. It takes three arguments: a *test-result* in level 3, a *true-clause* object in level 2, and a *false-clause* object in level 1. See "Conditional Examples" on page 1-23.

To enter IFTE in a program or in an algebraic:

- Press PRG BRCH NEXT ◀ IFTE.

The CASE ... END Structure

The syntax for this structure is

```
« ... CASE
      test-clause1 THEN true-clause1 END
      test-clause2 THEN true-clause2 END
      ⋮
      test-clausen THEN true-clausen END
      default-clause (optional)
    END ... »
```

The CASE ... END structure lets you execute a series of *test-clause* commands, then execute the appropriate *true-clause* sequence of commands. The first test that returns a true result causes execution of the corresponding true-clause, ending the CASE ... END structure. Optionally, you can include after the last test a *default-clause* that's executed if all the tests evaluate to false. If a test-clause is an algebraic, it's automatically evaluated to a number—you don't need →NUM or EVAL.

When CASE is executed, test-clause$_1$ is evaluated. If the test is true, true-clause$_1$ is executed, and execution skips to END. If test-clause$_1$ is false, execution proceeds to test-clause$_2$. Execution within the CASE structure continues until a true-clause is executed, or until all the test-clauses evaluate to false. If a default clause is included, it's

executed if all the test-clauses evaluate to false. See "Conditional Examples" below.

To enter CASE ... END in a program:

1. Press PRG BRCH ◀ CASE to enter CASE ... THEN ... END ... END.
2. For each additional test-clause, move the cursor after a test-clause END and press ▶ CASE to enter THEN ... END.

Conditional Examples

These examples illustrate conditional structures in programs.

Example: One Conditional Action. The programs below test the value in level 1—if the value is positive, it's made negative. The first program uses a command sequence as the test-clause:

```
« DUP IF 0 > THEN NEG END »
```

The value on the stack must be duplicated because the > command removes two arguments from the stack (0 and the copy of the value made by DUP).

The following version uses an algebraic as the test clause:

```
« + x « x IF 'x>0' THEN NEG END » »
```

The following version uses the IFT command:

```
« DUP 0 > « NEG » IFT »
```

Example: One Conditional Action. This program multiplies two numbers if both are nonzero.

Program:	**Comments:**
`« → x y`	Creates local variables x and y containing the two numbers from the stack.
`  « IF`	Starts the test-clause.
`  'x≠0'`	Tests one of the numbers and leaves a test result on the stack.
`  'y≠0'`	Tests the other number, leaving another test result on the stack.
`  AND`	Tests whether both tests were true.
`THEN`	Ends the test-clause, starts the true-clause.
`  x y *`	Multiplies the two numbers together only if AND returns true.
`  END`	Ends the true-clause.
`  »`	
`»`	

The following program accomplishes the same task as the previous program:

```
« → x y « IF 'x AND y' THEN x y * END » »
```

The test-clause `'x AND y'` returns "true" if both numbers are nonzero.

The following version uses the IFT command:

```
« → x y « 'x AND y' 'x*y' IFT » »
```

Example: Two Conditional Actions. This program takes a value x from the stack and calculates $(\sin x)/x$. At $x = 0$ the division would error, so the program returns the limit value 1 in this case.

```
« → x « IF 'x≠0' THEN x SIN x / ELSE 1 END » »
```

The following version uses IFTE algebraic syntax:

```
« → x 'IFTE(x≠0,SIN(x)/x,1)' »
```

Example: Two Conditional Actions. This program multiplies two numbers together if they're both nonzero—otherwise, it returns the string "ZERO".

Program:

Comments:

```
«
 → n1 n2                          Creates the local variables.
 «                                Starts the defining procedure.
  IF                              Starts the test clause.
   'n1≠0 AND n2≠0'                Tests n1 and n2.
  THEN                            If both numbers are nonzero,
   n1 n2 *                        multiplies the two values.
  ELSE                            Otherwise, returns the string
   "ZERO"                         ZERO.
  END                             Ends the conditional.
 »                                Ends the defining procedure.
»
```

Example: Two Conditional Actions. This program tests if two numbers on the stack have the same value. If so, it drops one of the numbers and stores the other in variable V1—otherwise, it stores the number from level 1 in V1 and the number from level 2 in V2.

Program:	**Comments:**
«	
IF	For the test clause, copies the
DUP2	numbers in levels 1 and 2 and
SAME	tests if they have the same value.
THEN	For the true clause, drops one of
DROP	the numbers and stores the other
'V1' STO	in *V1*.
ELSE	For the false clause, stores the
'V1' STO	level 1 number in *V1* and the
'V2' STO	level 2 number in *V2*.
END	Ends the conditional structure.
»	
ENTER	Puts the program on the stack.
' TST STO	Stores it in *TST*.

Enter the numbers 26 and 52, then execute *TST* to compare their values. Because the two number aren't equal, the VAR menu now contains two new variables *V1* and *V2*.

26 ENTER 52
VAR TST



Example: Multiple Conditional Actions. The following program stores the level 1 argument in a variable if the argument is a string, list, or program.

Program:	**Comments:**
≪	
÷ ⊔	Defines local variable y.
«	Starts the defining procedure.
CASE	Starts the case structure.
y TYPE 2 SAME	Case 1: If the argument is
THEN y 'STR' STO END	a string, stores it in STR.
y TYPE 5 SAME	Case 2: If the argument is
THEN y 'LIST' STO END	a list, stores it in $LIST$.
y TYPE 8 SAME	Case 3: If the argument is
THEN y 'PROG' STO END	a program, stores it in $PROG$.
END	Ends the case structure.
»	Ends the defining procedure.
»	

Using Loop Structures

You can use loop structures to execute a part of a program repeatedly. To specify in advance how many times to repeat the loop, use a *definite loop*. To use a test to determine whether or not to repeat the loop, use an *indefinite loop*.

Loop structures let a program execute a sequence of commands several times. Loop structures are built with commands—called structure words—that work only when used in proper combination with each other. These loop structure commands are contained in the PRG BRCH menu (PRG BRCH):

- START ... NEXT and START ... STEP.
- FOR ... NEXT and FOR ... STEP.
- DO ... UNTIL ... END.
- WHILE ... REPEAT ... END.

In addition, the Σ function provides an alternative to definite loop structures for summations.

Using Definite Loop Structures

Each of the two definite loop structures has two variations:

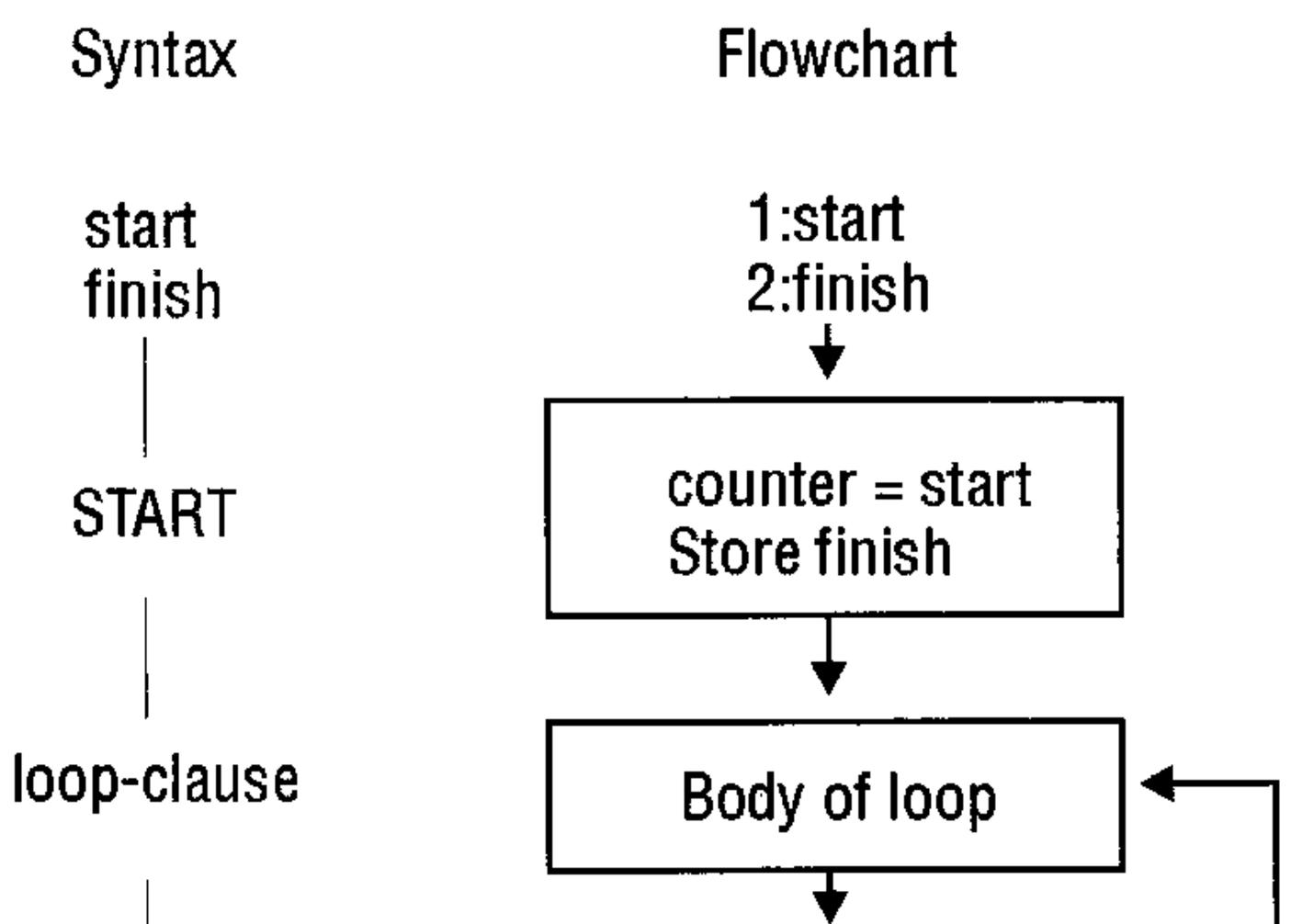
- NEXT. The counter increases by 1 for each loop.
- STEP. The counter increases or decreases by a specified amount for each loop.

The START ... NEXT Structure

The syntax for this structure is

※ ... *start finish* START *loop-clause* NEXT ... ※

START ... NEXT executes the *loop-clause* sequence of commands one time for each number in the range *start* to *finish*. The loop-clause is always executed at least once.



START ... NEXT Structure

START takes two numbers (*start* and *finish*) from the stack and stores them as the starting and ending values for a loop counter. Then, the loop-clause is executed. NEXT increments the counter by 1 and tests to see if its value is less than or equal to *finish*. If so, the loop-clause is executed again—otherwise, execution resumes following NEXT.

To enter START ... NEXT in a program:

- Press PRG BRCH ◀ START.

Example: The following program creates a list containing 10 copies of the string "ABC":

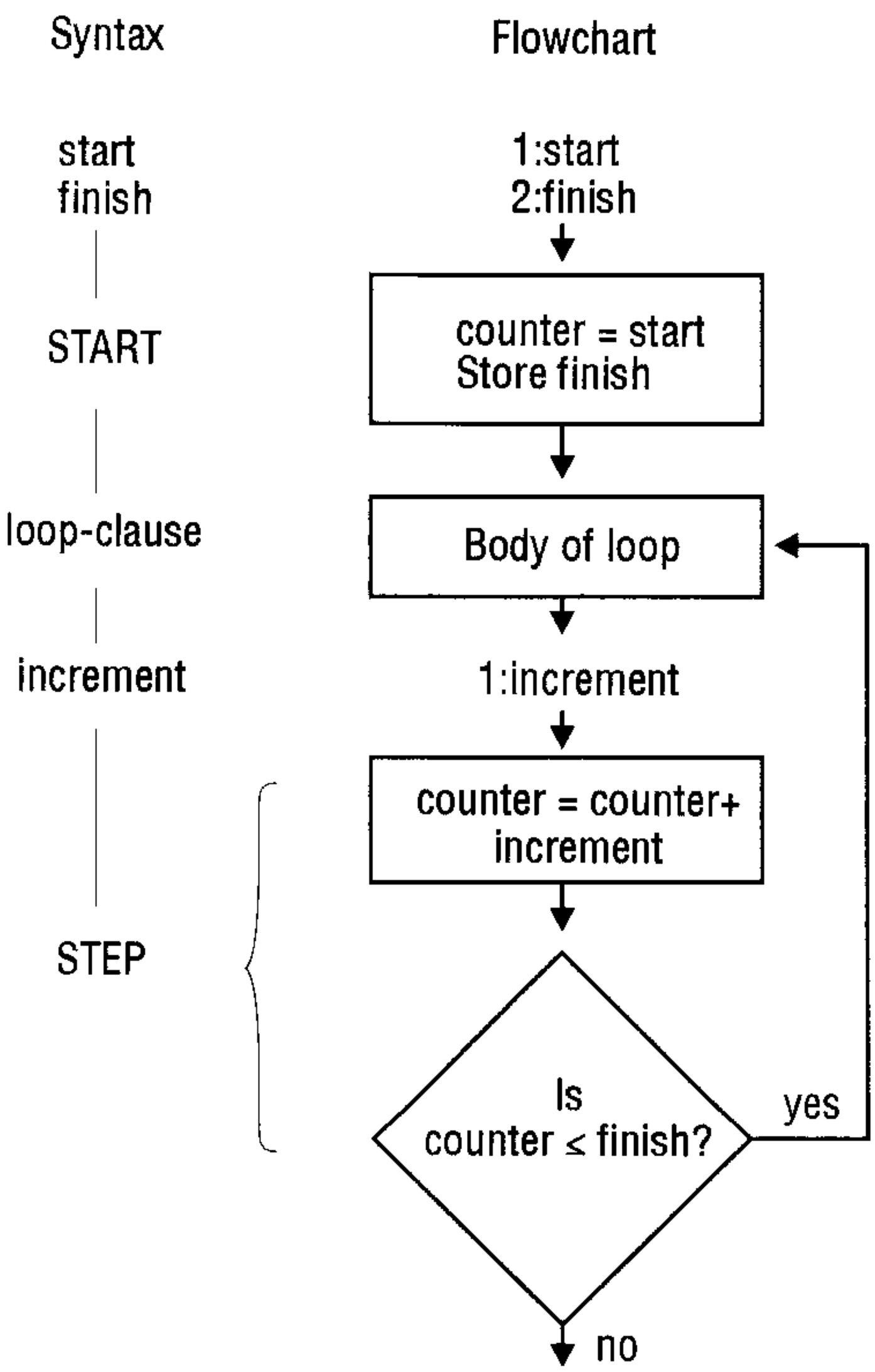
```
« 1 10 START "ABC" NEXT 10 →LIST »
```

The START ... STEP Structure

The syntax for this structure is

※ ... *start finish* START *loop-clause increment* STEP ... ※

START ... STEP executes the *loop-clause* sequence just like START ... NEXT does—except that the program specifies the increment value for the counter, rather than incrementing by 1. The loop-clause is always executed at least once.



START ... STEP Structure

START takes two numbers (*start* and *finish*) from the stack and stores them as the starting and ending values of the loop counter. Then the loop-clause is executed. STEP takes the increment value from the stack and increments the counter by that value. If the argument

of STEP is an algebraic or a name, it's automatically evaluated to a number.

The increment value can be positive or negative. If it's positive, the loop is executed again if the counter is less than or equal to *finish*. If the increment value is negative, the loop is executed if the counter is greater than or equal to *finish*. Otherwise, execution resumes following STEP. In the previous flowchart, the increment value is positive.

To enter START ... STEP in a program:

- Press PRG BRCH ▶ START.

Example: The following program takes a number x from the stack and calculates the square of that number several times ($x/3$ times):

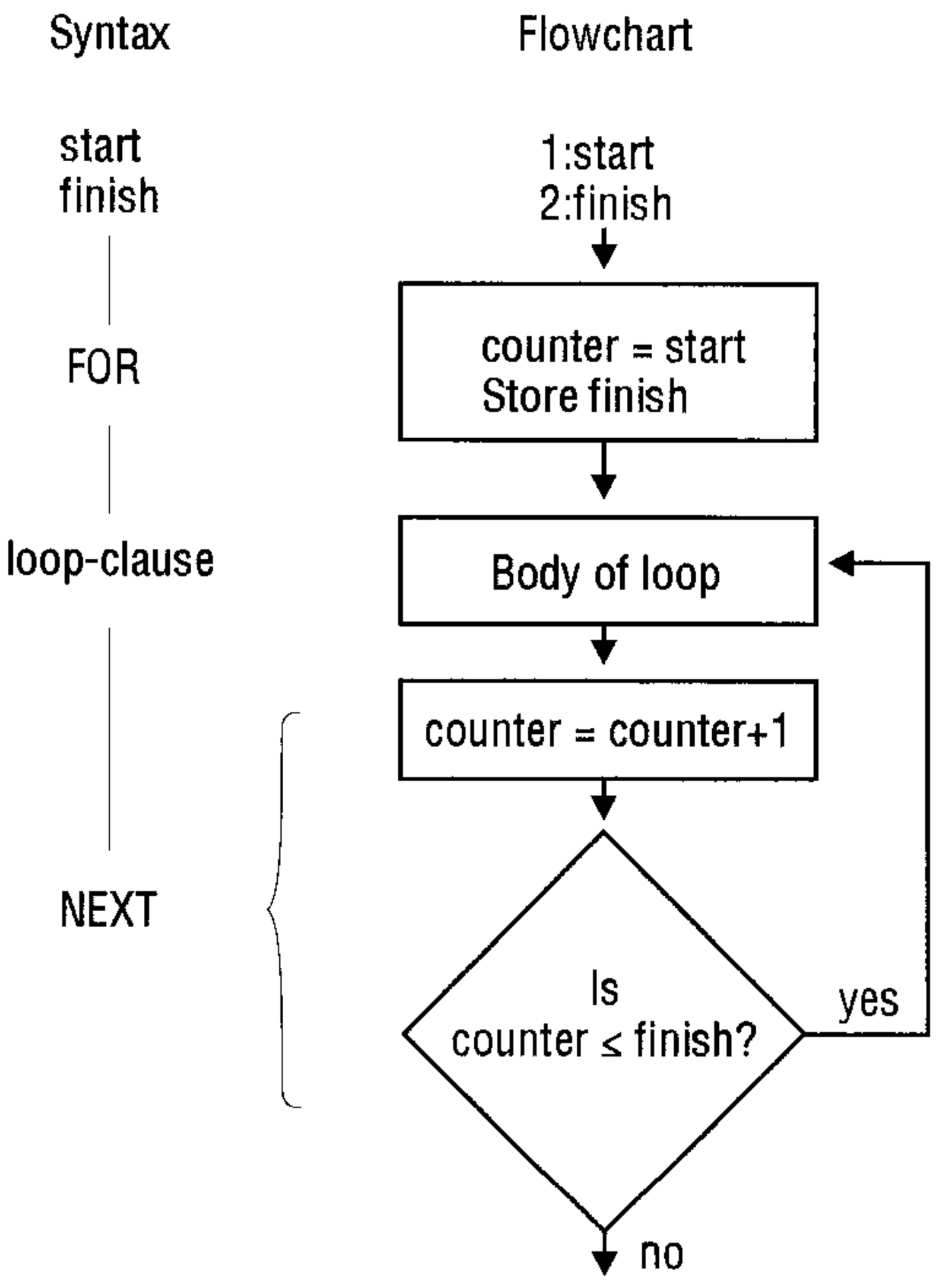
```
« DUP → x « x 1 START x SQ -3 STEP » »
```

The FOR ... NEXT Structure

The syntax for this structure is

« ... *start finish* FOR *counter loop-clause* NEXT ... »

FOR ... NEXT executes the *loop-clause* program segment one time for each number in the range *start* to *finish*, using local variable *counter* as the loop counter. You can use this variable in the loop-clause. The loop-clause is always executed at least once.



FOR ... NEXT Structure

FOR takes *start* and *finish* from the stack as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Then the loop-clause is executed—*counter* can appear within the loop-clause. NEXT increments *counter-name* by one, and then tests whether its value is less than or equal to *finish*. If so, the *loop-clause* is repeated (with the new value of *counter*)—otherwise,

execution resumes following NEXT. When the loop is exited, *counter* is purged.

To enter FOR ... NEXT in a program:

- Press PRG BRCH ↩ FOR.

Example: The following program places the squares of the integers 1 through 5 on the stack:

```
« 1 5 FOR j j SQ NEXT »
```

Example: The following program takes the value x from the stack and computes the integer powers i of x. For example, when $x = 12$ and *start* and *finish* are 3 and 5 respectively, the program returns 12^3, 12^4, and 12^5. It requires as inputs *start* and *finish* in levels 3 and 2, and x in level 1. (→ × removes x from the stack, leaving *start* and *finish* there as arguments for FOR.)

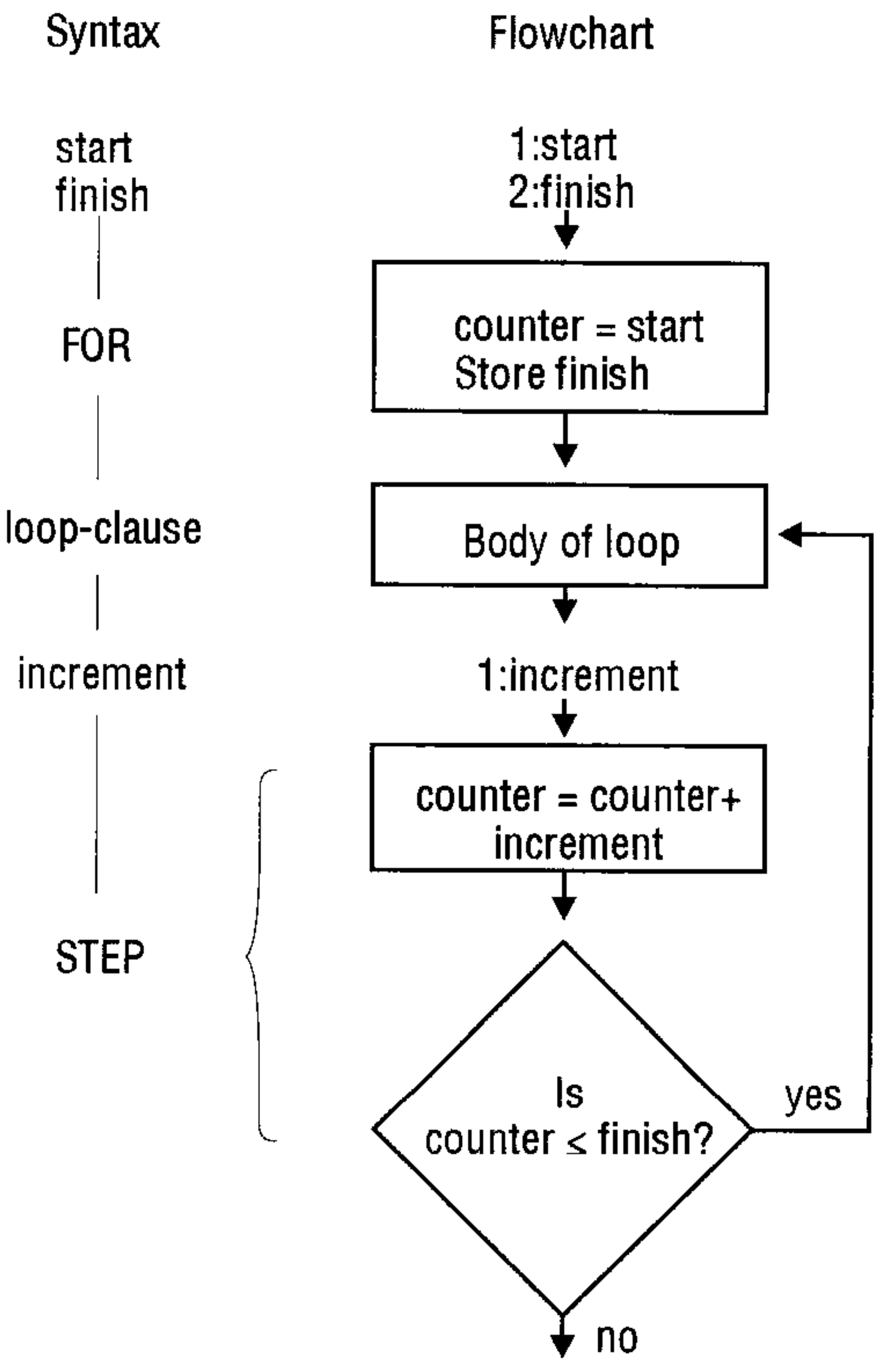
```
« → x « FOR n 'x^n' EVAL NEXT » »
```

The FOR ... STEP Structure

The syntax for this structure is

※ *... start finish* FOR *counter loop-clause increment* STEP *...* ※

FOR ... STEP executes the *loop-clause* sequence just like FOR ... NEXT does—except that the program specifies the increment value for *counter*, rather than incrementing by 1. The loop-clause is always executed at least once.



FOR ... STEP Structure

FOR takes *start* and *finish* from the stack as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Next, the loop-clause is executed—*counter* can appear within the loop-clause. STEP takes the increment value from the

stack and increments *counter* by that value. If the argument of STEP is an algebraic or a name, it's automatically evaluated to a number.

The increment value can be positive or negative. If the increment is positive, the loop is executed again if *counter* is less than or equal to *finish*. If the increment is negative, the loop is executed if *counter* is greater than or equal to *finish*. Otherwise, *counter* is purged and execution resumes following STEP. In the previous flowchart, the increment value is positive.

To enter FOR ... STEP in a program:

- Press PRG BRCH ▶ FOR.

Example: The following program places the squares of the integers 1, 3, 5, 7, and 9 on the stack:

```
« 1 9 FOR x x SQ 2 STEP »
```

Example: The following program takes *n* from the stack, and returns the series of numbers 1, 2, 4, 8, 16, ... , *n*. If *n* isn't in the series, the program stops at the last value less than *n*.

```
« 1 SWAP FOR n n n STEP »
```

The first n is the local variable declaration for the FOR loop. The second n is put on the stack each iteration of the loop. The third n is used by STEP as the step increment.

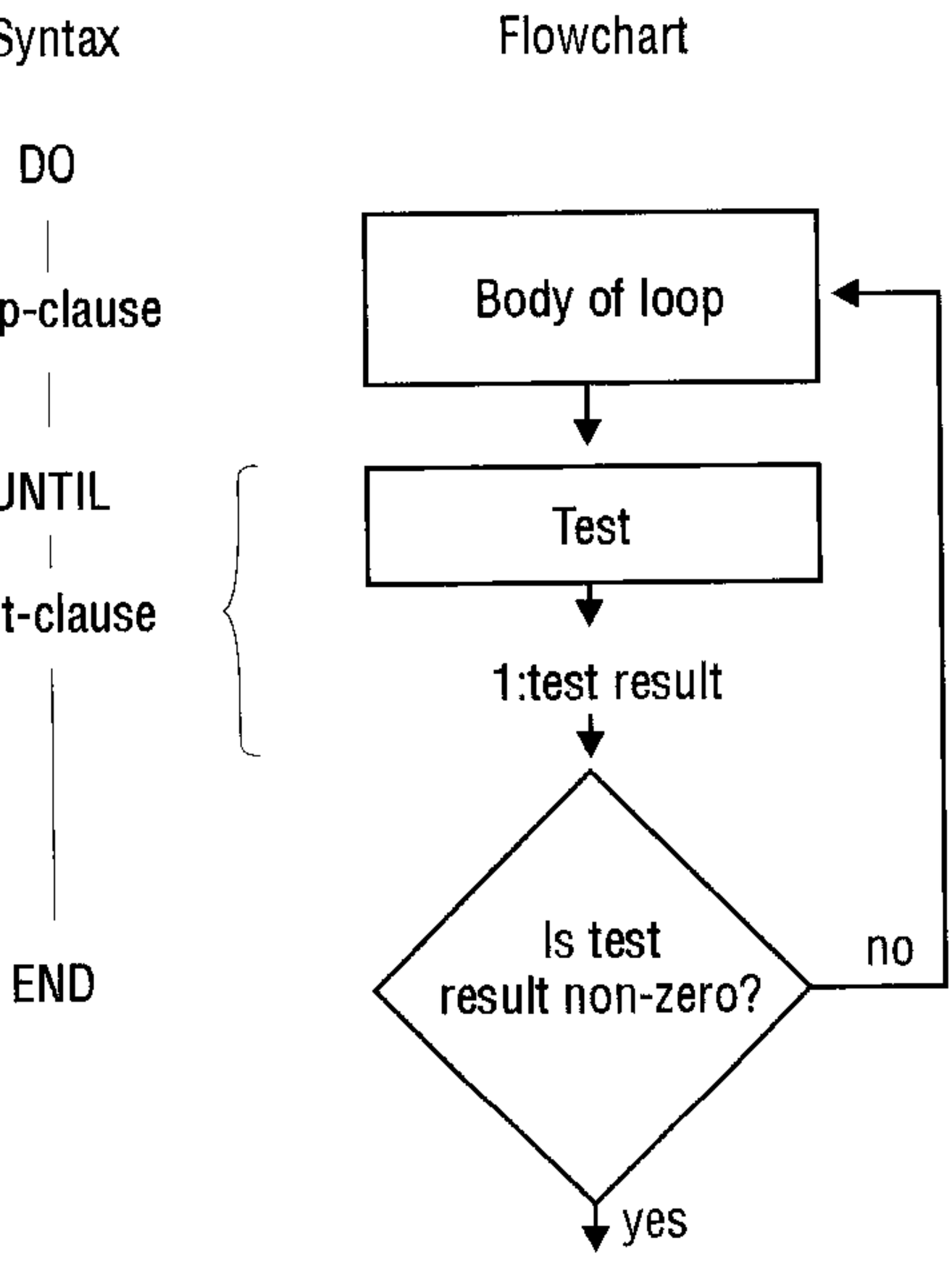
Using Indefinite Loop Structures

The DO ... UNTIL ... END Structure

The syntax for this structure is

« ... DO *loop-clause* UNTIL *test-clause* END ... »

DO ... UNTIL ... END executes the *loop-clause* sequence repeatedly until *test-clause* returns a true (nonzero) result. Because the test-clause is executed *after* the loop-clause, the loop-clause is always executed at least once.



DO ... UNTIL ... END Structure

DO starts execution of the loop-clause. UNTIL marks the end of the loop-clause. The test-clause leaves a test result on the stack. END removes the test result from the stack. If its value is zero, the loop-clause is executed again—otherwise, execution resumes following END. If the argument of END is an algebraic or a name, it's automatically evaluated to a number.

To enter DO ... UNTIL ... END in a program:

■ Press (PRG) BRCH (◀) DO.

Example: The following program calculates $n + 2n + 3n + \ldots$ for a value ot n. The program stops when the sum exceeds 1000, and returns the sum and the coefficient of n.

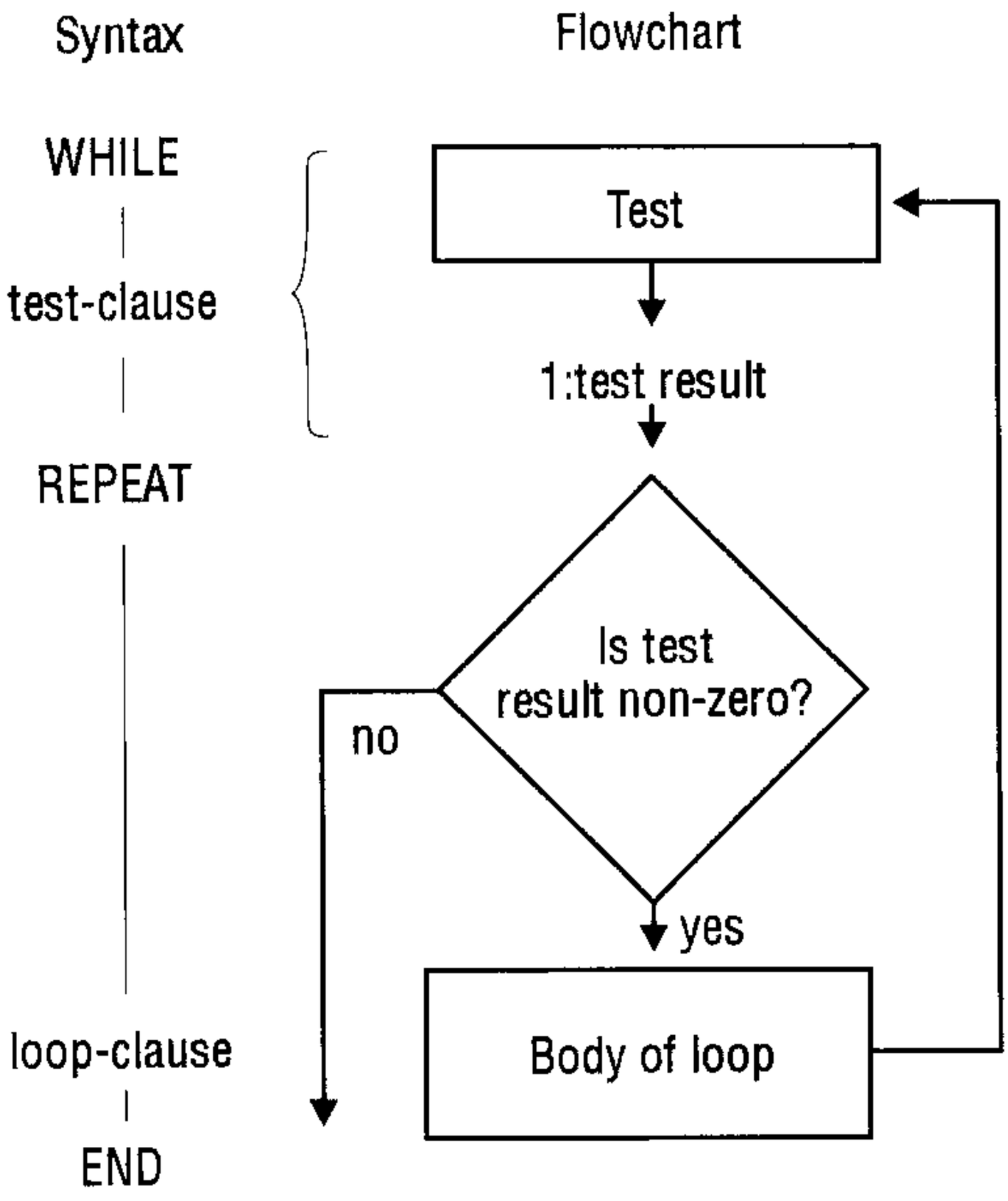
Program:	**Comments:**
«	
DUP 1	Duplicates n, stores the value into n and s, and initializes c to 1.
→ n s c	Starts the defining procedure.
«	Starts the loop-clause.
DO	Increments the counter by 1. (See "Using Loop Counters" on page 1-39.)
'c' INCR	Calculates $c \times n$ and adds the product to s.
n * 's' STO+	Starts the test clause.
UNTIL	Repeats loop until $s > 1000$.
s 1000 >	Ends the test-clause.
END	Puts s and c on the stack.
s c	Ends the defining procedure.
»	
»	

The WHILE ... REPEAT ... END Structure

The syntax for this structure is

« ... WHILE *test-clause* REPEAT *loop-clause* END ... »

WHILE ... REPEAT ... END repeatedly evaluates *test-clause* and executes the *loop-clause* sequence if the test is true. Because the test-clause is executed *before* the loop-clause, the loop-clause is not executed if the test is initially false.



WHILE ... REPEAT ... END Structure

WHILE starts execution of the test-clause, which returns a test result to the stack. REPEAT takes the value from the stack. If the value is nonzero, execution continues with the loop-clause—otherwise, execution resumes following END. If the argument of REPEAT is an algebraic or a name, it's automatically evaluated to a number.

To enter WHILE ... REPEAT ... END in a program:

- Press PRG BRCH ◀ WHILE.

Example: The following program starts with a number on the stack, and repeatedly performs a division by 2 as long as the result is evenly divisible. For example, starting with the number 24, the program computes 12, then 6, then 3.

```
« WHILE DUP 2 MOD 0 == REPEAT 2 / DUP END DROP »
```

Example: The following program takes any number of vectors or arrays from the stack and adds them to the statistics matrix. (The vectors and arrays must have the same number of columns.) WHILE ... REPEAT ... END is used instead of DO ... UNTIL ... END because the test must be done *before* the addition. (If *only* vectors or arrays with the same number of columns are on the stack, the program errors after the last vector or array is added to the statistics matrix.)

```
« WHILE DUP TYPE 3 == REPEAT Σ+ END »
```

Using Loop Counters

For certain problems you may need a counter inside a loop structure to keep track of the number of loops. (This counter isn't related to the counter variable in a FOR ... NEXT/STEP structure.) You can use any global or local variable as a counter. You can use the INCR or DECR command to increment or decrement the counter value *and* put its new value on the stack.

The syntax for INCR and DECR is

« ... '*variable*' INCR ... »

or

« ... '*variable*' DECR ... »

To enter INCR or DECR in a program:

- Press ◀ MEMORY ARITH INCR or DECR.

The INCR and DECR commands take a global or local variable name (with ' delimiters) as their argument—the variable must contain a real number. The command does the following:

1. Changes the value stored in the variable by +1 or −1.
2. Returns the new value to the stack.

Examples: If c contains the value 5, then `'C' INCR` stores 6 in c and returns 6 to the stack.

The following program takes a maximum of five vectors from the stack and adds them to the current statistics matrix.

Program:	**Comments:**
`«`	
`0 → C`	Stores 0 in local variable c.
`«`	Starts the defining procedure.
`WHILE`	Starts the test clause.
`DUP TYPE 3 ==`	Returns true if level 1 contains a vector.
`'C' INCR`	Increments and returns the value in c.
`5 ≤`	Returns true if the counter $c \leq 5$.
`AND`	Returns true if the two previous test results are true.
`REPEAT`	
`Σ+`	Adds the vector to ΣDAT.
`END`	Ends the structure.
`»`	Ends the defining procedure.
`»`	

Using Summations Instead of Loops

For certain calculations that involve summations, you can use the Σ function instead of loops. You can use Σ with stack syntax or with algebraic syntax. Σ automatically repeats the addition for the specified range of the index variable—without using a loop structure.

Example: The following programs take an integer upper limit n from the stack, then find the summation

$$\sum_{j=1}^{n} j^2$$

One program uses a FOR ... NEXT loop—the other uses Σ.

Program:

```
«
 0 1 ROT

 FOR j
  j SQ +
 NEXT
»
```

Comments:

Initializes the summation and puts the limits in place.
Loops through the calculation.

Program:

```
«
 → n
 'Σ(j=1,n,j^2)'
»
```

Comments:

Uses Σ to calculate the summation.

Example: The following program uses ΣLIST to calculate the summation of all elements of a vector or matrix. The program takes from the stack an array or a name that evaluates to an array, and returns the summation.

Program:

```
«
 OBJ→
 1
 +
 ΠLIST

 →LIST
 ΣLIST
»
```

Comments:

Finds the dimensions of the array and leaves it in a list on level 1.
Adds 1 to the list. (If the array is a vector, the list on level 1 has only one element. ΠLIST will error if the list has fewer than two elements.)
Multiplies all of the list elements together.
Converts the array elements into a list, and sums them.

Using Flags

You can use flags to control calculator behavior and program execution. You can think of a flag as a switch that is either on (*set*) or off (*clear*). You can test a flag's state within a conditional or loop structure to make a decision. Because certain flags have unique meanings for the calculator, flag tests expand a program's decision-making capabilities beyond that available with comparison and logical functions.

Types of Flags

The HP 48 has two types of flags:

- **System flags.** Flags −1 through −64. These flags have predefined meanings for the calculator.
- **User flags.** Flags 1 through 64. User flags are not used by any built-in operations. What they mean depends entirely on how the *program* uses them.

Appendix C lists the 64 system flags and their definitions. For example, system flag −40 controls the clock display—when this flag is *clear* (the default state), the clock is not displayed—when this flag is *set*, the clock is displayed. (When you press CLK in the ◀ MODES MISC menu, you are setting or clearing flag −40.)

When you set user flag 1 through 5, the corresponding annunciator is turned on. Certain plug-in cards may use user-flags in the range 31 through 64.

Setting, Clearing, and Testing Flags

Flag commands take a flag number from the stack—an integer 1 through 64 (for user flags) or −1 through −64 (for system flags).

To set, clear, or test a flag:

1. Enter the flag number (positive or negative).
2. Execute the flag command—see the table below.

Flag Commands

Key	Programmable Command	Description
PRG TEST NXT NXT or ◀ MODES FLAG:		
SF	SF	Sets the flag.
CF	CF	Clears the flag.
F S ?	FS?	Returns 1 (true) if the flag is set, or 0 (false) if the flag is clear.
F C ?	FC?	Returns 1 (true) if the flag is clear, or 0 (false) if the flag is set.
F S ? C	FS?C	Tests the flag (returns true if the flag is set), then clears the flag.
F C ? C	FC?C	Tests the flag (returns true if the flag is clear), then clears the flag.

Example: System Flag. The following program sets an alarm for June 6, 1993 at 5:05 PM. It first tests the status of system flag −42 (Date Format flag) in a conditional structure and then supplies the alarm date in the current date format, based on the test result.

Program:

Comments:

```
«
  IF                          Tests the status of flag −42, the
   -42 FC?                    Date Format flag.
  THEN                        If flag −42 is clear, supplies the
   6.151993                   date in month/day/year format.
  ELSE                        If flag −42 is set, supplies the
   15.061993                  date in day.month.year format.
  END                         Ends the conditional.
  17.05 "TEST COMPLETE"       Sets the alarm: 17.05 is the alarm
  3 →LIST STOALARM            time and "TEST COMPLETE"
                              is the alarm message.
»
```

Example: User Flag. The following program returns either the fractional or integer part of the number in level 1, depending on the state of user flag 10.

Program:

```
«
 IF
  10 FS?
 THEN
  IP
 ELSE
  FP
 END
»
```

Comments:

Starts the conditional.
Tests the status of user flag 10.
If flag 10 is set, returns the integer part.
If flag 10 is clear, returns the fractional part.
Ends the conditional.

To use this program, you enter a number, either set flag 10 (to get the integer part) or clear flag 10 (to get the fractional part), then run the program.

Recalling and Storing the Flag States

If you have a program that changes the state of a flag during execution, you may want it to save and restore original flag states.

The RCLF (recall flags) and STOF (store flags) commands let you recall and store the states of the HP 48 flags. For these commands, a 64-bit binary integer represents the states of 64 flags—each 0 bit corresponds to a flag that's clear, each 1 bit corresponds to a flag that's set. The rightmost (least significant) bit corresponds to system flag −1 or user flag 1.

To recall the current flag states:

- Execute RCLF (◀ MODES FLAG NXT RCLF).

RCLF returns a list containing two 64-bit binary integers representing the current states of the system and user flags:

{ #n_s #n_u }

To change the current flag states:

1. Enter the flag-state argument—see below.
2. Execute STOF (◀ MODES FLAG NXT STOF).

STOF sets the current states of flags based on the flag-state argument:

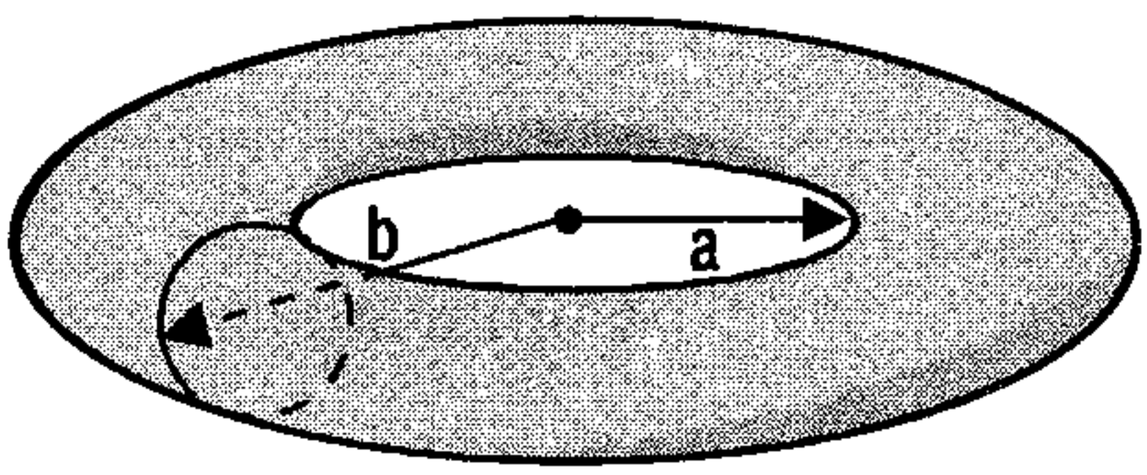
$\#n_{\mathrm{s}}$	Changes the states of only the system flags.
{ $\#n_{\mathrm{s}}$ $\#n_{\mathrm{u}}$ }	Changes the states of the system and user flags.

Example: The program *PRESERVE* on page 2-8 uses RCLF and STOF.

Using Subroutines

Because a program is itself an object, it can be used in another program as a subroutine. When program B is used by program A, program A *calls* program B, and program B is a *subroutine* in program A.

Example: The program *TORSA* calculates the surface area of a torus of inner radius a and outer radius b. *TORSA* is used as a subroutine in a second program *TORSV*, which calculates the volume of a torus.



The surface area and volume are calculated by

$$A = \pi^2(b^2 - a^2) \qquad V = \frac{1}{4}\pi^2(b^2 - a^2)(b - a)$$

(The quantity $\pi^2(b^2 - a^2)$ in the second equation is the surface area of a torus calculated by *TORSA*.)

Here are the stack diagram and program listing for *TORSA*.

Level 2	Level 1	→	Level 1
a	*b*	→	*surface area*

Program:

```
≪
  → a b
  'π^2*(b^2-a^2)'
  →NUM
≫
```

Comments:

Creates local variables *a* and *b*.
Calculates the surface area.
Converts algebraic to a number.

ENTER
' TORSA STO

Puts the program on the stack.
Stores the program in *TORSA*.

Here is a stack diagram and program listing for *TORSV*.

Level 2	Level 1	→	Level 1
a	*b*	→	*volume*

Program:

```
≪
  → a b
  ≪
    a b TORSA
    b a - * 4 /
  ≫
≫
```

Comments:

Creates local variables *a* and *b*.
Starts a program as the defining procedure.
Puts the numbers stored in *a* and *b* on the stack, then calls *TORSA* with those arguments.
Completes the volume calculation using the surface area.
Ends the defining procedure.

ENTER
' TORSV STO

Puts the program on the stack.
Stores the program in *TORSV*.

Now use *TORSV* to calculate the volume of a torus of inner radius $a = 6$ and outer radius $b = 8$.

6 ENTER 8
VAR TORSV



Single-Stepping through a Program

It's easier to understand how a program works if you execute it step by step, observing the effect of each step. Doing this can help you debug your own programs or understand programs written by others.

To single-step from the start of a program:

1. Put the program or program name in level 1 (or the command line).
2. Press PRG NXT RUN DBUG to start and immediately suspend execution.

   HALT appears in the status area.
3. Take any action:
   - To see the next program step displayed in the status area and then executed, press SST.
   - To display but not execute the next one or two program steps, press NEXT.
   - To continue with normal execution, press ↩ CONT.
   - To abandon further execution, press KILL.
4. Repeat the previous step as desired.

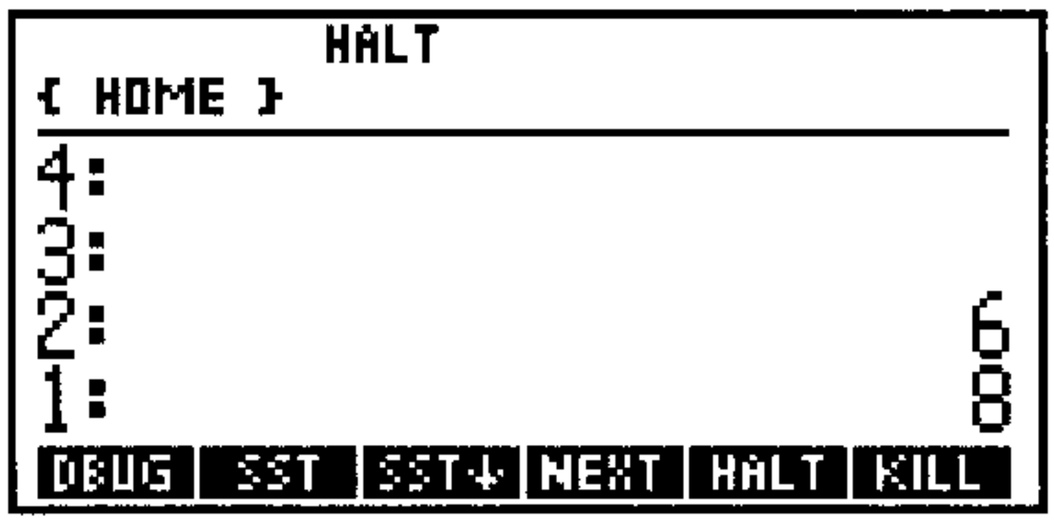
To turn off the HALT annunciator at any time:

- Press PRG NXT RUN KILL.

Example: Execute program *TORSV* step by step. Use $a = 6$ and $b = 8$.

Select the VAR menu and enter the data. Enter the program name and start the debugging. HALT indicates program execution is suspended.

◀ CLEAR
VAR
6 ENTER 8 ENTER
▽ TOOLS
PRG NXT RUN DEBUG



Display and execute the first program step. Notice that it takes the two arguments from the stack and stored them in local variables *a* and *b*.

SST



Continue single-stepping until the status area shows the current directory. Watch the stack and status area as you single-step through the program.

SST ... SST



To single-step from the middle of a program:

1. Insert a HALT command in the program where you want to begin single-stepping.
2. Execute the program normally. The program stops when the HALT command is executed, and the HALT annunciator appears.
3. Take any action:
   - To see the next program step displayed in the status area and then executed, press SST.
   - To display but not execute the next one or two program steps, press NEXT.
   - To continue with normal execution, press ◀ CONT.
   - To abandon further execution, press KILL.
4. Repeat the previous step as desired.

When you want the program to run normally again, remove the HALT command from the program.

To single-step when the next step is a subroutine:

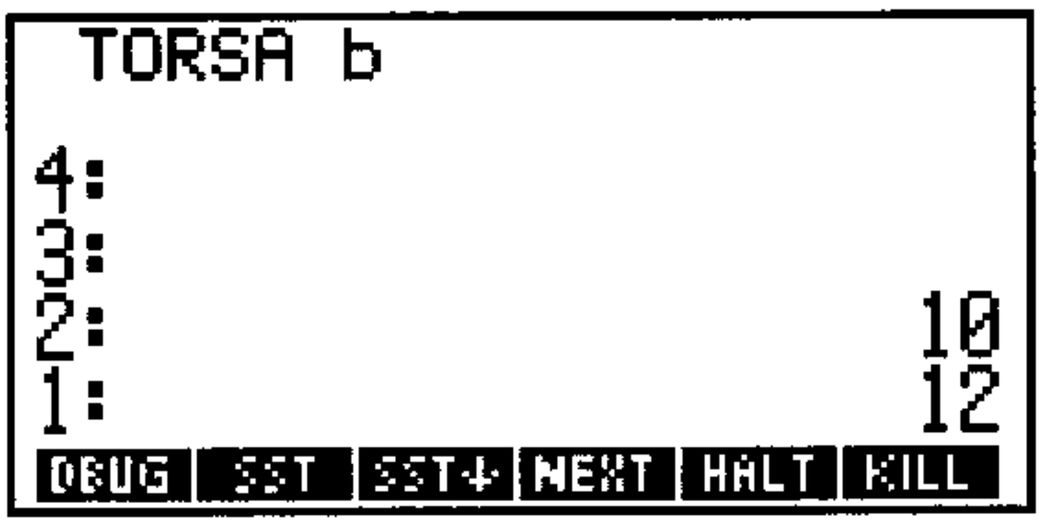
- To execute the subroutine in one step, press SST.
- To execute the subroutine step-by-step, press SST↓.

SST executes the next step in a program—if the next step is a subroutine, SST executes the subroutine in one step. SST↓ works just like SST—except if the next program step is a subroutine, it single-steps to the first step in the subroutine.

Example: In the previous example, you used SST to execute subroutine *TORSA* in one step. Now execute program *TORSV* step by step to calculate the volume of a torus of radii $a = 10$ and $b = 12$. When you reach subroutine *TORSA*, execute it step by step.

Select the VAR menu and enter the data. Enter the program name and start the debugging. Execute the first four steps of the program, then check the next step.

◀ CLEAR VAR
10 ENTER 12
' TORSV
PRG NXT RUN DEBUG
SST↓ (4 times)
NEXT



The next step is TORSA. Single-step into *TORSA*, then check that you're at the first step of *TORSA*.

SST↓
NEXT



Press ◀ CONT ◀ CONT to complete subroutine and program execution.

The following table summarizes the operations for single-stepping through a program.

Single-Step Operations

Key	Programmable Command	Description
PRG NXT RUN:		
DEBUG		Starts program execution, then suspends it as if HALT were the first program command. Takes as its argument the program or program name in level 1.
SST		Executes the next object or command in the suspended program.
SST↓		Same as SST, except if the next program step is a subroutine, single-steps to the first step in that subroutine.
NEXT		Displays the next one or two objects, but does not execute them. The display persists until the next keystroke.
HALT	HALT	Suspends program execution at the location of the HALT command in the program.
KILL	KILL	Cancels all suspended programs and turns off the HALT annunciator.
← CONT	CONT	Resumes execution of a halted program.

Trapping Errors

If you attempt an invalid operation from the keyboard, the operation is not executed and an error message appears. For example, if you execute + with a vector and a real number on the stack, the HP 48 returns the message `+ Error: Bad Argument Type` and returns the arguments to the stack (if Last Arguments is enabled).

In a program, the same thing happens, but program execution is also aborted. If you anticipate error conditions, your program can process them without interrupting execution.

For simple programs, you can run the program again if it stops with an error. For other programs, you can design them to *trap* errors and continue executing. You can also create user-defined errors to trap certain conditions in programs. The error trapping commands are located in the PRG ERROR menu.

Causing and Analyzing Errors

Many conditions are automatically recognized by the HP 48 as error conditions—and they're automatically treated as errors in programs.

You can also define conditions that cause errors. You can cause a *user-defined error* (with a user-defined error message)—or you can cause a built-in error. Normally, you'll include a conditional or loop structure with a test for the error condition—and if it occurs, you'll cause the user-defined or built-in error to occur.

To cause a user-defined error to occur in a program:

1. Enter a string (with " " delimiters) containing the desired error message.
2. Enter the DOERR command (PRG ERROR menu).

To artificially cause a built-in error to occur in a program:

1. Enter the error number (as a binary integer or real number) for the error.
2. Enter the DOERR command (PRG ERROR menu).

If DOERR is trapped in an IFERR structure (described in the next topic), execution continues. If it's not trapped, execution is abandoned at the DOERR command and the error message appears.

To analyze an error in a program:

- To get the error number for the last error, execute ERRN (PRG ERROR menu).
- To get the error message for the last error, execute ERRM (PRG ERROR menu).
- To clear the last-error information, execute ERR0 (PRG ERROR menu).

The error number for a user-defined error is #70000h. See the list of built-in error numbers in appendix A, "Error and Status Messages."

Example: The following program aborts execution if the list in level 1 contains three objects.

```
«
 OBJ→
 IF 3 SAME
 THEN "3 OBJECTS IN LIST" DOERR
 END
»
```

The following table summarizes error trapping commands.

Error Trapping Commands

Key	Programmable Command	Description
PRG NXT ERROR:		
DOERR	DOERR	Causes an error. For a string in level 1, causes a user-defined error: the calculator behaves just as if an ordinary error has occurred. For a binary integer or real number in level 1, causes the corresponding built-in error. If the error isn't trapped in an IFERR structure, DOERR displays the message and abandons program execution. (For 0 in level 1, abandons execution without updating the error number or message—like CANCEL.)
ERRN	ERRN	Returns the error number, as a binary integer, of the most recent error. Returns #0 if the error number was cleared by ERR0.
ERRM	ERRM	Returns the error message (a string) for the most recent error. Returns an empty string if the error number was cleared by ERR0.
ERR0	ERR0	Clears the last error number and message.

Making an Error Trap

You can construct an error trap with one of the following conditional structures:

- IFERR ... THEN ... END.
- IFERR ... THEN ... ELSE ... END.

The IFERR ... THEN ... END Structure

The syntax for this structure is

« ... IFERR *trap-clause* THEN *error-clause* END ... »

The commands in the error-clause are executed only if an error is generated during execution of the trap-clause. If an error occurs in the trap-clause, the error is ignored, the remainder of the trap-clause is skipped, and program execution jumps to the error-clause. If *no* errors occur in the trap-clause, the error-clause is skipped and execution resumes after the END command.

To enter IFERR ... THEN ... END in a program:

- Press PRG NXT ERROR ◀ IFERR.

Example: The following program takes any number of vectors or arrays from the stack and adds them to the statistics matrix. However, the program stops with an error if a vector or array with a different number of columns is encountered. In addition, if *only* vectors or arrays with the same number of columns are on the stack, the program stops with an error after the last vector or array has been removed from the stack.

```
« WHILE DUP TYPE 3 == REPEAT Σ+ END »
```

In the following revised version, the program simply attempts to add the level 1 object to the statistics matrix until an error occurs. Then, it ends by displaying the message DONE.

Program:

```
«
 IFERR
  WHILE
   1
  REPEAT
   Σ+
  END
 THEN
  "DONE" 1 DISP
  1 FREEZE

 END
»
```

Comments:

Starts the trap-clause.
The WHILE structure repeats indefinitely, adding the vectors and arrays to the statistics matrix until an error occurs.

Starts the error clause. If an error occurs in the WHILE structure, displays the message DONE in the status area.
Ends the error structure.

The IFERR ... THEN ... ELSE ... END Structure

The syntax for this structure is

« ... IFERR *trap-clause*
THEN *error-clause* ELSE *normal-clause* END ... »

The commands in the error-clause are executed only if an error is generated during execution of the trap-clause. If an error occurs in the trap-clause, the error is ignored, the remainder of the trap-clause is skipped, and program execution jumps to the error-clause. If *no* errors occur in the trap-clause, execution jumps to the normal-clause at the completion of the trap-clause.

To enter IFERR ... THEN ... ELSE ... END in a program:

- Press PRG NXT ERROR ▶ IFERR.

Example: The following program prompts for two numbers, then adds them. If only one number is supplied, the program displays an error message and prompts again.

Program:	Comments:
«	
DO	Begins the main loop.
"KEY IN a AND b" " "	Prompts for two numbers.
INPUT OBJ→	
UNTIL	Starts the loop test clause.
IFERR	The error trap contains only the
+	+ command.
THEN	If an error occurs, recalls and
ERRM 5 DISP	displays the `Too Few Arguments` message for 2
2 WAIT	seconds, then puts 0 (false) on
0	the stack for the main loop.
ELSE	If no error occurs, puts 1 (true)
1	on the stack for the main loop.
END	Ends the error trap.
END	Ends the main loop. If the error trap left 0 (false) on the stack, the main loop repeats—otherwise, the program ends.
»	

Input

A program can stop for user input, then resume execution, or can use choose boxes or input forms (dialog boxes) for input. You can use several commands to get input:

- PROMPT (◀ CONT to resume).
- DISP FREEZE HALT (◀ CONT to resume).
- INPUT (ENTER to resume).
- INFORM
- CHOOSE

Data Input Commands

Key	Command	Description
PRG NXT IN:		
INFOR	INFORM	Creates a user-defined input form.
NOVA	NOVAL	Place holder for the INFORM command. Returned when a value is not present in an input form field.
CHOOS	CHOOSE	Creates a user-defined choose box.
KEY	KEY	Returns a test result to level 1 and, if a key is pressed, the location of that key (level 2).
WAIT	WAIT	Suspends program execution for a specified duration (in seconds, level 1).
INPUT	INPUT	Suspends program execution for data input.
PROM	PROMPT	Halts program execution for data input.

Using PROMPT ... CONT for Input

PROMPT uses the status area for prompting, and allows the user to use normal keyboard operations during input.

To enter PROMPT in a program:

1. Enter a string (with " " delimiters) to be displayed as a prompt in the status area.
2. Enter the PROMPT command (PRG IN menu).

※ ... "*prompt-string*" PROMPT ... ※

PROMPT takes a string argument from level 1, displays the string (without the " " delimiters) in the status area, and halts program execution. Calculator control is returned to the keyboard.

When execution resumes, the input is left on the stack as entered.

To respond to PROMPT while running a program:

1. Enter your input—you can use keyboard operations to calculate the input.

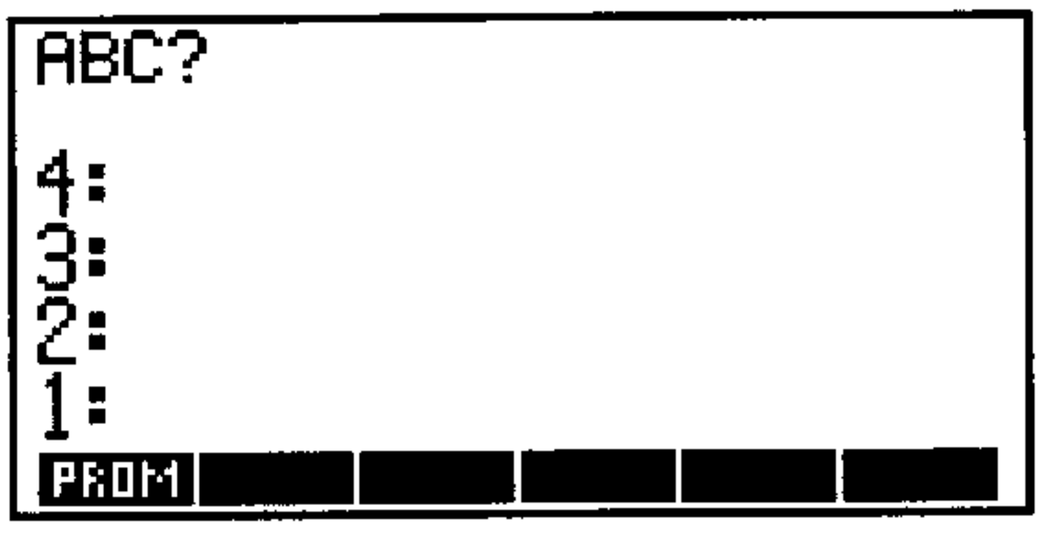
2. Press ◀ CONT.

The message remains until you press ENTER or CANCEL or until you update the status area.

Example: If you execute this program segment

```
« "ABC?" PROMPT »
```

the display looks like this:



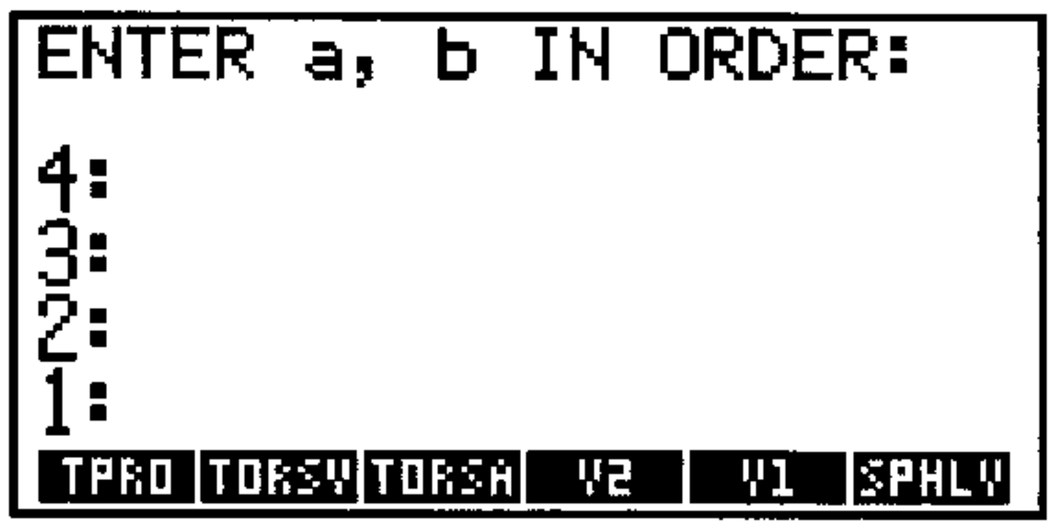
Example: The following program, *TPROMPT*, prompts you for the dimensions of a torus, then calls program *TORSA* (from page 1-45) to calculate its surface area. You don't have to enter data on the stack prior to program execution.

Program:	**Comments:**
«	
"ENTER a, b IN ORDER:"	Puts the prompting string on the stack.
PROMPT	Displays the string in the status area, halts program execution, and returns calculator control to the keyboard.
TORSA	Executes *TORSA* using the just-entered stack arguments.
»	
ENTER ' TPROMPT STO	Stores the program in *TPROMPT*.

Execute *TPROMPT* to calculate the volume of a torus with inner radius $a = 8$ and outer radius $b = 10$.

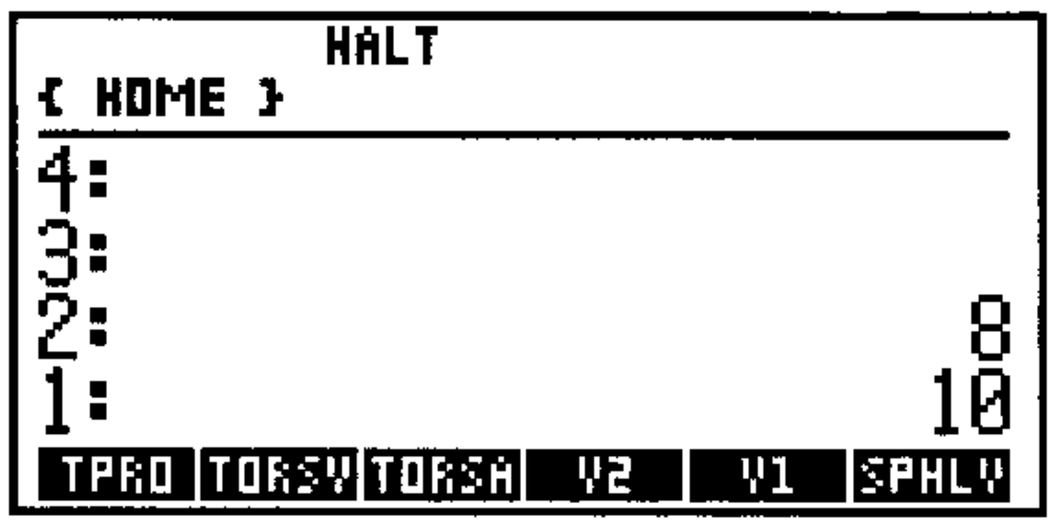
Execute *TPROMPT*. The program prompts you for data.

◀ CLEAR
VAR TPRO



Enter the inner and outer radii. After you press ENTER, the prompt message is cleared from the status area.

8 ENTER 10



Continue the program.

◀ CONT



Note that when program execution is suspended by PROMPT, you can execute calculator operations just as you did before you started the program. If the outer radius b of the torus in the previous example is measured as 0.83 feet, you can convert that value to inches *while the program is suspended for data input* by pressing .83 ENTER 12 ×, then ◀ CONT.

Using DISP FREEZE HALT ... CONT for Input

DISP FREEZE HALT lets you control the entire display during input, and allows the user to use normal keyboard operations during input.

To enter DISP FREEZE HALT in a program:

1. Enter a string or other object to be displayed as a prompt.
2. Enter a number specifying the line to display it on.
3. Enter the DISP command (PRG OUT menu).
4. Enter a number specifying the areas of the display to "freeze."
5. Enter the FREEZE command (PRG OUT menu).
6. Enter the HALT command (PRG OUT menu).

« ... *prompt-object display-line* DISP
freeze-area FREEZE HALT ... »

DISP displays an object in a specified line of the display. DISP takes two arguments from the stack: an object from level 2, and a display-line number 1 through 7 from level 1. If the object is a string, it's displayed without the " " delimiters. The display created by DISP persists only as long as the program continues execution—if the program ends or is suspended by HALT, the calculator returns to the normal stack environment and updates the display. However, you can use FREEZE to retain the prompt display.

FREEZE "freezes" display areas so they aren't updated until a *key press*. Argument *n* in level 1 is the sum of the codes for the areas to be frozen: 1 for the status area, 2 for the stack/command line area, 4 for the menu area.

HALT suspends program execution at the location of the HALT command and turns on the HALT annunciator. Calculator control is returned to the keyboard for normal operations.

When execution resumes, the input remains on the stack as entered.

To respond to HALT while running a program:

1. Enter your input—you can use keyboard operations to calculate the input.
2. Press ↩ CONT.

Example: If you execute this program segment

« "ABC▪DEF▪GHI" CLLCD 1 DISP 3 FREEZE HALT »

the display looks like this:



(The ▪ in the previous program is the calculator's representation for the ↵ newline character after you enter a program on the stack.)

Using INPUT ... ENTER for Input

INPUT lets you use the stack area for prompting, lets you supply default input, and prevents the user from using normal stack operations or altering data on the stack.

To enter INPUT in a program:

1. Enter a string (with " " delimiters) to be displayed as a prompt at the top of the stack area.
2. Enter a string or list (with delimiters) that specifies the command-line content and behavior—see below.
3. Enter the INPUT command (PRG IN menu).
4. Enter OBJ→ (PRG TYPE menu) or other command that processes the input as a string object.

« ... "*prompt-string*" "*command-line*" INPUT OBJ→ ... »

or

« ... "*prompt-string*" {*command-line*} INPUT OBJ→ ... »

INPUT, in its simplest form, takes two strings as arguments—see the list of additional options following. INPUT blanks the stack area, displays the contents of the level-2 string at the top of the stack area, and displays the contents of the level-1 string in the command line. It then activates Program-entry mode, puts the insert cursor after the string in the command line, and suspends execution.

When execution resumes, the input is returned to level 1 as a string object, called the *result string*.

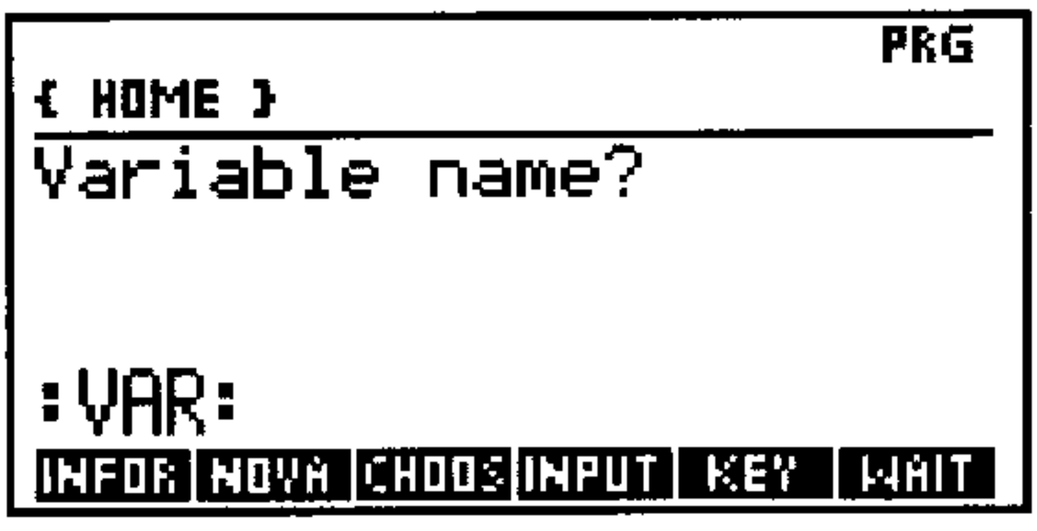
To respond to INPUT while running a program:

1. Enter your input. (You can't execute commands—they're simply echoed in the command line.)
2. Optional: To clear the command line and start over, press CANCEL.
3. Press ENTER.

Example: If you execute this program segment

```
« "Variable name?" ":VAR:" INPUT »
```

the display looks like this:

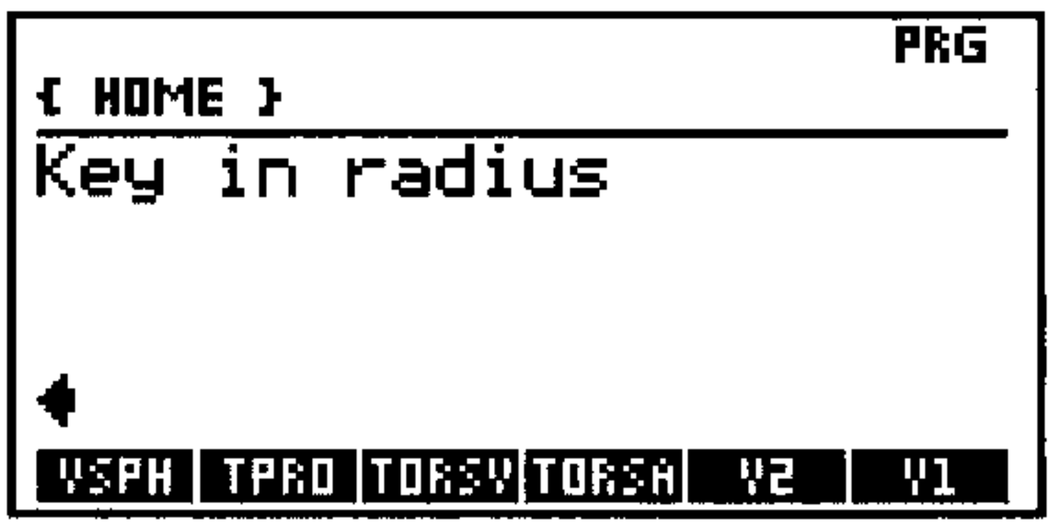


Example: The following program, *VSPH*, calculates the volume of a sphere. *VSPH* prompts for the radius of the sphere, then cubes it and multiplies by $^4/_3\ \pi$. *VSPH* executes INPUT to prompt for the radius. INPUT sets Program-entry mode when program execution pauses for data entry.

Program:	**Comments:**
«	
"Key in radius"	Specifies the prompt string.
""	Specifies the command-line string. In this case, the command line will be empty.
INPUT	Displays the prompt, puts the cursor at the start of the command line, and suspends the program for data input (the radius of the sphere).
OBJ→	Converts the result string into its component object—a real number.
3 ^	Cubes the radius.
4 * 3 / π * →NUM	Completes the calculation.
»	
ENTER ' VSPH STO	Stores the program in *VSPH*.

Execute *VSPH* to calculate the volume of a sphere of radius 2.5.

(VAR) ¥VSPH



Key in the radius and continue program execution.

2.5 (ENTER)



To include INPUT options:

- Use a list (with { } delimiters) as the command-line argument for INPUT. The list can contain one or more of the following:
  - Command-line string (with " " delimiters).
  - Cursor position as a real number or as a list containing two real numbers.
  - Operating options ALG, α, or ¥.

In its general form, the level 1 argument for INPUT is a list that specifies the content and interpretation of the command line. The list can contain one or more of the following parameters in any order:

{ "*command-line*" *cursor-position operating-options* }

"*command-line*"	Specifies the content of the command line when the program pauses. Embedded newline characters produce multiple lines in the display. (If not included, the command line is blank.)
cursor-position	Specifies the position of the cursor in the command line and its type. (If not included, an insert cursor is at the end of the command line.) ■ A *real number n* specifies the *n*th character in the first row (line) of the command line. Zero specifies the end of the command-line string. A positive number specifies the *insert* cursor—a negative number specifies the *replace* cursor. ■ A *list* {*row character*} specifies the row and character position. Row 1 is the first row (line) of the command line. Characters count from

the left end of each row—character 0 specifies the end of the row. A positive row number specifies the *insert* cursor—a negative row number specifies the *replace* cursor.

operating-options Specify the input setup and processing using zero or more of these unquoted names:

- ALG activates Algebraic/Program-entry mode (for algebraic syntax). (If not included, Program-entry mode is active.)
- α (α A) specifies alpha lock. (If not included, alpha is inactive.)
- √ verifies whether the result string (without the " " delimiters) is a valid object or sequence of objects. If the result string isn't valid, INPUT displays the `Invalid Syntax` message and prompts again for data. (If not included, syntax isn't checked.)

To design the command-line string for INPUT:

- For simple input, use a string that produces a valid object:
  - Use an empty string.
  - Use a :*label*: tag.
  - Use a "*text*" comment.
- For special input, use a string that produces a recognizable pattern.

After the user enters input in the command line and presses ENTER to resume execution, the contents of the command line are returned to level 1 as the result string. The result string normally contains the original command-line string, too. If you design the command-line string carefully, you can ease the process of extracting the input data.

To process the result string from INPUT:

- For simple input, use OBJ→ to convert the string into its corresponding objects.
- For sensitive input, use the √ option for INPUT to check for valid objects, then use OBJ→ to convert the string into those objects.
- For special input, process the input as a string object, possibly extracting data as substrings.

Example: The program *VSPH* on page 1-61 uses an empty command-line string.

Example: The program *SSEC* on page 1-66 uses a command-line string whose characters form a pattern. The program extracts substrings from the result string.

Example: The command-line string "@UPPER LIMIT@" displays @UPPER LIMIT@# in the command line. If you press 200 ENTER, the return string is "@UPPER LIMIT@200". When OBJ→ extracts the text from the string, it strips away the @ characters and the enclosed characters, and it returns the number 200. (See "Creating Programs on a Computer" on page 1-10 for more information about @ comments.)

Example: The following program, *TINPUT*, executes INPUT to prompt for the inner and outer radii of a torus, then calls *TORSA* (page 1-45) to calculate its surface area. *TINPUT* prompts for a and b in a two-row command line. The level 1 argument for INPUT is a list that contains:

- The command-line string, which forms the tags and delimiters for two tagged objects.
- An embedded list specifying the initial cursor position.
- The ∀ parameter to check for invalid syntax in the result string.

Program:

Comments:

```
«
  "Key in a, b"

  { "▪a▪■▪b▪" {1 0} V }

  INPUT

  OBJ→

  TORSA

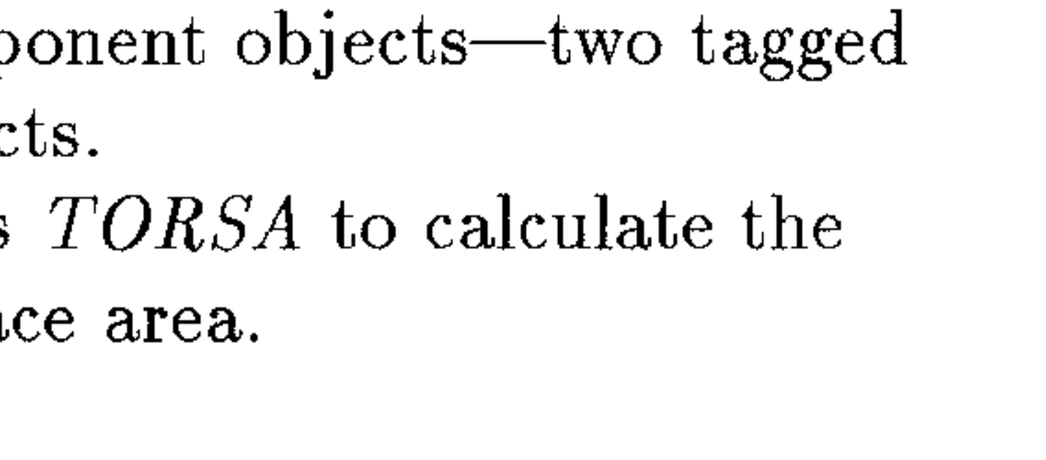
»
```

The level 2 string, displayed at the top of the stack area.
The level 1 list contains a string, a list, and the verify option. (To key in the string, press ▶ " " ▶ : : a ▶ ▶ ↵ ▶ : : b. After you press ENTER to put the finished program on the stack, the string is shown on one line, with ■ indicating the newline character.) The embedded list puts the insert cursor at the end of row 1.
Displays the stack and command-line strings, positions the cursor, sets Program-entry mode, and suspends execution for input.
Converts the string into its component objects—two tagged objects.
Calls *TORSA* to calculate the surface area.

ENTER ' TINPUT STO Stores the program in *TINPUT*.

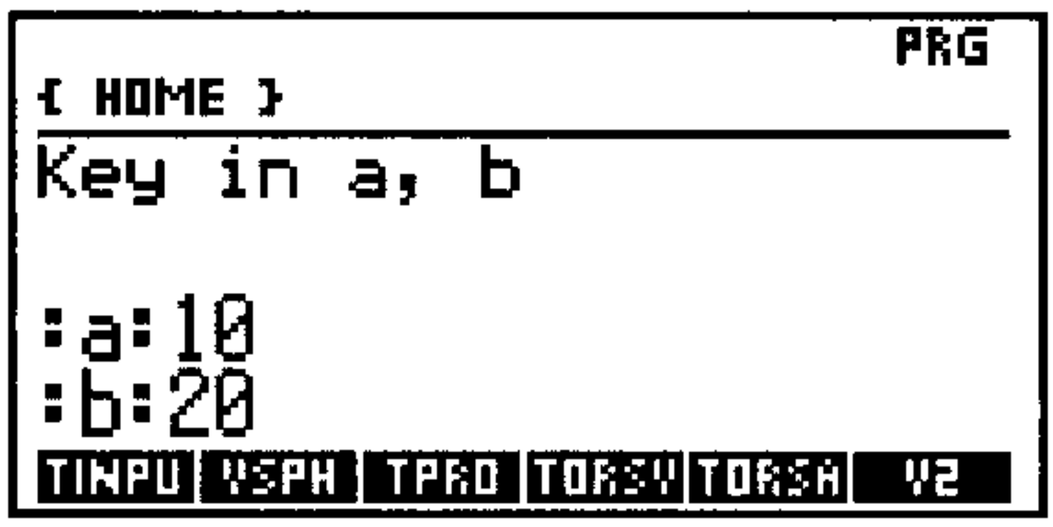
Execute *TINPUT* to calculate the surface area of a torus of inner radius $a = 10$ and outer radius $b = 20$.

VAR TINPU



Key in the value for a, press ▼ to move the cursor to the next prompt, then key in the value for b.

10 ▼ 20



Continue program execution.

ENTER



Example: The following program executes INPUT to prompt for a social security number, then extracts two strings: the first three digits and last four digits. The level 1 argument for INPUT specifies:

- A command-line string with dashes.
- The *replace* cursor positioned at the start of the prompt string (−1). This lets the user "fill in" the command line string, using ▶ to skip over the dashes in the pattern.
- By default, no verification of object syntax—the dashes make the content invalid as objects.

Level 1	→	Level 2	Level 1
	→	*"last four digits"*	*"first three digits"*

Program:	**Comments:**
«	
"Key in S.S. #"	Prompt string.
{ " - - " -1 }	Command-line string (3 spaces before the first −, 2 spaces between, and 4 spaces after the last −).
INPUT	Suspends the program for input.
DUP 1 3 SUB	Copies the result string, then
SWAP	extracts the first three and last
8 11 SUB	four digits in string form.
»	
ENTER ' SSEC STO	Stores the program in *SSEC*.

Using INFORM and CHOOSE for Input

You can use input forms (dialog boxes), and choose boxes for program input. Programs that contain input forms or choose boxes wait until you acknowledge them (OK or CANCEL) before they continue execution.

If OK is pressed, CHOOSE returns the selected item (or its designated returned value) to level 2 and a 1 to level 1. INFORM returns a list of field values to level 2 and a 1 to level 1.

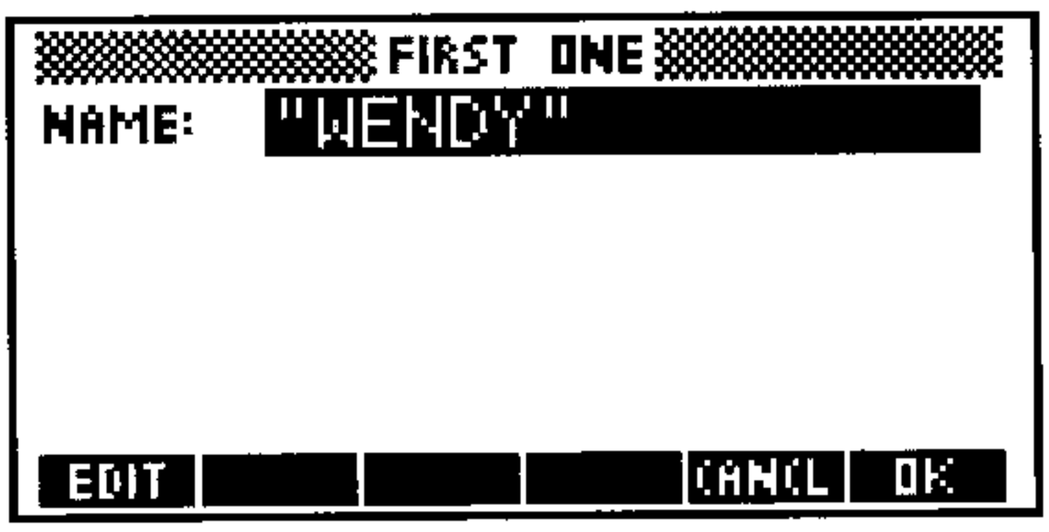
Both the INFORM and CHOOSE commands return 0 if CANCEL is pressed.

To set up an input form:

1. Enter a title string for the input form (use → " ").
2. Enter a list of field specifications.
3. Enter a list of format options.
4. Enter a list of reset values (values that appear when RESET is pressed).
5. Enter a list of default values.
6. Execute the INFORM command.

Example: Enter a title "FIRST ONE" ENTER.
Specify a field { "Name:" } ENTER.
Enter format options (one column, tabs stop width five) { 1 5 } ENTER.

Enter reset value for the field { "THERESA" } ENTER.
Enter default value for the field { "WENDY" } ENTER.
Execute INFORM (PRG NXT IN INFOR).
The screen on the left appears. Press NXT RESET OK and the screen on the right appears.



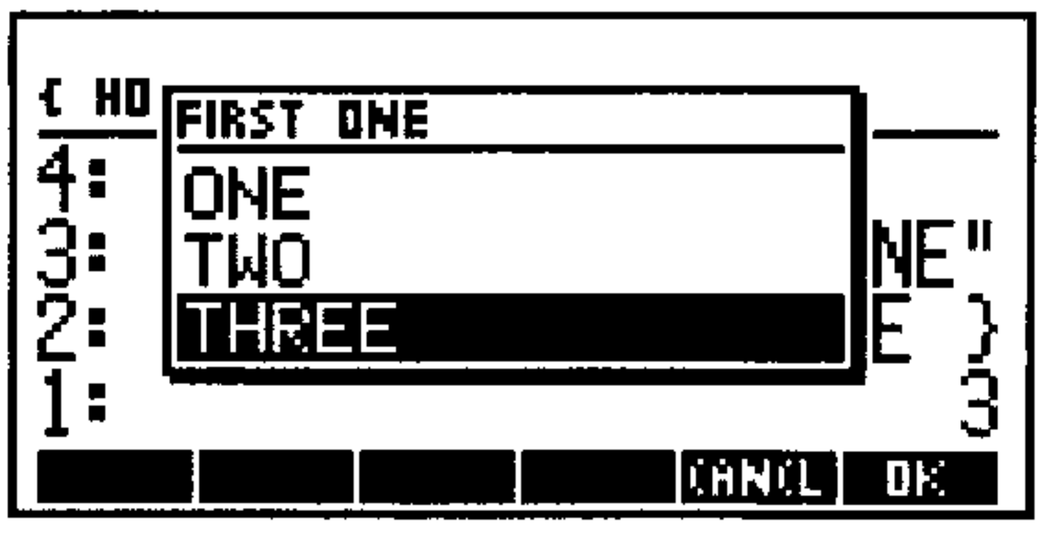


You can specify a help message and the type of data that must be entered in a field by entering field specifications as lists. For example, { {"Name:" "Enter your name" 2} } defines the Name field, displays Enter your name across the bottom of the input form, and accepts only object type 2 (strings) as input.

To set up a choose box:

1. Enter a title string for the choose box.
2. Enter a list of items. If this is a list of two-element lists, the first element is displayed in the choose box, and the *second* element is returned to level 2 when OK is pressed.
3. Enter a position number for the default highlighted item. (0 makes a view-only choose box.)
4. Execute the CHOOSE command.

Example: Enter a title "FIRST ONE" ENTER.
Enter a list of items { ONE TWO THREE } ENTER.
Enter a position number for default highlighted value 3 ENTER.
Execute CHOOSE (PRG NXT IN CHOOS).
The following choose box appears:



Example: The following program uses input forms, choose boxes, and message boxes to create a simple phone list database.

Program:

```
«
'NAMES' VTYPE
   IF -1 ==
   THEN { } 'NAMES' STO
   END
 WHILE
"PHONELIST OPTIONS:"
{
{ "ADD A NAME" 1 }
{ "VIEW A NUMBER" 2 }
 } 1 CHOOSE
  REPEAT → c «
CASE c 1 ==
        THEN
           WHILE
```

Comments:

Checks if the name list (NAMES) exists, if not, creates an empty one.

While cancel is not pressed, creates a choose box that lists the database options. When OK is pressed, the second item in the list pair is returned to the stack.

Stores the returned value in *c*.

Case 1 (ADD name), while cancel is not pressed, do the following:

Program:

```
"ADD A NAME"
 {
 { "NAME:" "ENTER NAME" 2 }
 { "PHONE:" "ENTER A
PHONE NUMBER" 2 } }
 { } { } { } INFORM
 REPEAT
DUP
  IF { NOVAL } HEAD POS
  THEN
   DROP
   "Complete both fields
   before pressing OK"
   MSGBOX
 ELSE 1
   +LIST NAMES + SORT
   'NAMES' STO
   END
  END
 END
{ 2 ==
 THEN
  IF { } NAMES SAME
THEN
 "YOU MUST ADD A
   NAME FIRST"
  MSGBOX
```

Comments:

Creates an input form that gets the name and phone number. The two fields accept only strings (object type 2).

Checks if either field in the new entry is blank.

If either one is, displays a message.

If neither are, adds the list to NAMES, sorts it, and stores it back in NAMES. Ends the IF structure, the WHILE loop, and the CASE statement.

Case 2 (View a Number).

Checks if NAMES is an empty list.
If it is, displays a message.

Program:

```
 ELSE
    WHILE
     "VIEW A NUMBER"
     NAMES 1 CHOOSE
  REPEAT
     →STR MSGBOX
     END
   END
  END
END
※
END
※
```

ENTER ' PHONES STO

Comments:

If NAMES isn't empty, creates a choose box using NAMES as choice items.

When OK is pressed, the second item in the NAMES list pairs (the phone number) is returned. Makes it a string and displays it.

Ends the WHILE loop, the IF structure, and the CASE statement.

Ends the CASE structure, marks the end of the local variable defining procedure, ends the WHILE loop, and marks the end the program.

Stores the program in *PHONES*.

You can delete names and numbers by editing the NAMES variable. To improve upon this program, create a delete name routine.

Beeping to Get Attention

To enter BEEP in a program:

1. Enter a number that specifies the tone frequency in hertz.
2. Enter a number that specifies the tone duration in seconds.
3. Enter the BEEP command (PRG NXT ▒LIT▒ menu).

```
※ ... frequency duration BEEP ... ※
```

BEEP takes two arguments from the stack: the tone frequency from level 2 and the tone duration from level 1.

Example: The following edited version of *TPROMPT* sounds a 440-hertz, one-half-second tone at the prompt for data input.

Program:

```
«
 "ENTER a, b IN ORDER:"
 440 .5 BEEP

 PROMPT
 TORSA
»
```

Comments:

Sounds a tone just before the prompt for data input.

Stopping a Program for Keystroke Input

A program can stop for keystroke input—it can wait for the user to press a key. You can do this with the WAIT and KEY commands.

Using WAIT for Keystroke Input

The WAIT command normally suspends execution for a specified number of seconds. However, you can specify that it wait indefinitely until a key is pressed.

To enter WAIT in a program:

- To stop without changing the display, enter 0 and the WAIT command (PRG IN menu).
- To stop and display the current menu, enter −1 and the WAIT command (PRG IN menu).

WAIT takes the 0 or −1 from level 1, then suspends execution until a valid keystroke is executed.

For an argument of −1, WAIT displays the currently specified menu. This lets you build and display a menu of user choices while the program is paused. (A menu built with MENU or TMENU is not normally displayed until the program ends or is halted.)

When execution resumes, the three-digit key location number of the pressed key is left on the stack. This number indicates the row, column, and shift level of the key.

To respond to WAIT while running a program:

- Press any valid keystroke. (A prefix key such as ◀ or α by itself is not a valid keystroke.)

Using KEY for Keystroke Input

You can use KEY inside an indefinite loop to "pause" execution until any key—or a certain key—is pressed.

To enter a KEY loop in a program:

1. Enter the loop structure.
2. In the test-clause sequence, enter the KEY command (PRG IN menu) plus any necessary test commands.
3. In the loop-clause, enter *no* commands to give the appearance of a "paused" condition.

KEY returns 0 to level 1 when the loop begins. It continues to return 0 until a key is pressed—then it returns 1 to level 1 and the two-digit row-column number of the pressed key to level 2. For example, ENTER returns 51, and ◀ returns 71.

The test-clause should normally cause the loop to repeat until a key is pressed. If a key is pressed, you can use comparison tests to check the value of the key number. (See "Using Indefinite Loop Structures" on page 1-36 and "Using Comparison Functions" on page 1-17.)

To respond to a KEY loop while running a program:

- Press any key. (A prefix key such as ◀ or α *is* a valid key.)

Example: The following program segment returns 1 to level 1 if + is pressed, or 0 to level 1 if any other key is pressed:

```
« ... DO UNTIL KEY END 95 SAME ... »
```

Output

You can determine how a program presents its output. You can make the output more recognizable using the techniques described in this section.

Data Output Commands

Key	Command	Description
PRG NXT IO:		
PVIEW	PVIEW	Displays PICT starting at the given coordinates.
TEXT	TEXT	Displays the stack display.
CLLCD	CLLCD	Blanks the stack display.
DISP	DISP	Displays an object in the specified line.
FREEZ	FREEZE	"Freezes" a specified area of the display until a key press.
MSGB	MSGBOX	Creates a user-defined message box.
BEEP	BEEP	Sounds a beep at a specified frequency (in hertz, level 2) and duration (in seconds, level 1).

Labeling Output with Tags

To label a result with a tag:

1. Put the output object on the stack.
2. Enter a tag—a string, a quoted name, or a number.
3. Enter the →TAG command (PRG TYPE menu).

« ... *object tag* →TAG ... »

→TAG takes two arguments—an object and a tag—from the stack and returns a tagged object.

Example: The following program *TTAG* is identical to *TINPUT*, except that it returns the result as AREA: *value*.

Program:

```
«
 "Key in a, b"
 « ":a:■:b:" C1 0) V »
 INPUT OBJ→
 TORSA
 "AREA"
 →TAG
»
```

Comments:

Enters the tag (a string).
Uses the program result and string to create the tagged object.

(ENTER) (') TTAG (STO) — Stores the program in *TTAG*.

Execute *TTAG* to calculate the area of a torus of inner radius $a = 1.5$ and outer radius $b = 1.85$. The answer is returned as a tagged object.

(VAR) TTAG
1.5 (▼) 1.85
(ENTER)



Labeling and Displaying Output as Strings

To label and display a result as a string:

1. Put the output object on the stack.
2. Enter the →STR command (PRG TYPE menu).
3. Enter a string to label the object (with " " delimiters).
4. Enter the SWAP + commands to swap and concatenate the strings.
5. Enter a number specifying the line to display the string on.
6. Enter the DISP command (PRG OUT menu).

« ... *object* →STR *label* SWAP + *line* DISP ... »

DISP displays a string without its " " delimiters.

Example: The following program *TSTRING* is identical to *TINPUT*, except that it converts the program result to a string and appends a labeling string to it.

Program:

```
«
  "Key in a, b"
  { ":a:■:b:" (1 0) V }
  INPUT OBJ→
  TORSA
  →STR
  "Area = "
  SWAP +
  CLLCD 1 DISP 1 FREEZE
»
```

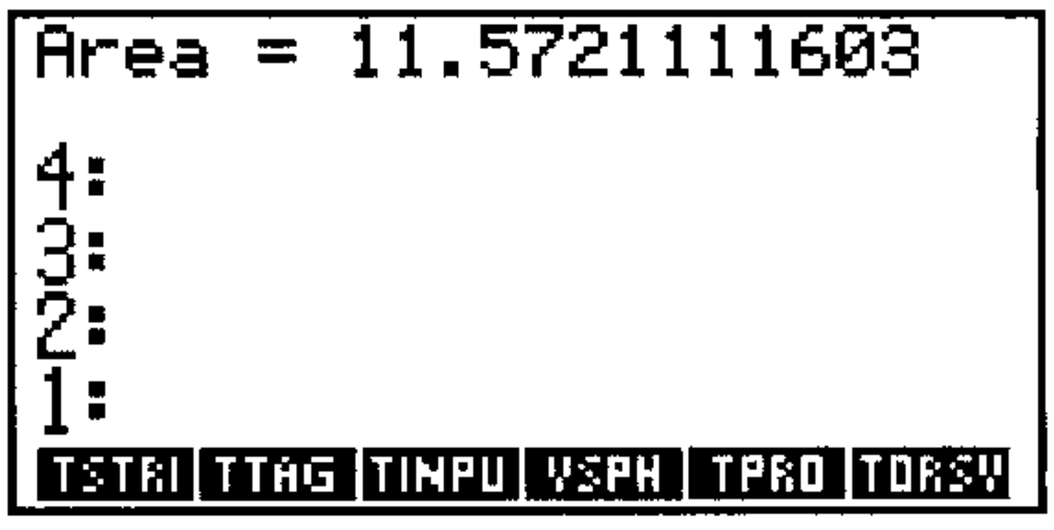
Comments:

Converts the result to a string.
Enters the labeling string.
Swaps and adds the two strings.
Displays the resultant string, without its delimiters, in line 1 of the display.

ENTER ' TSTRING STO — Stores the program in *TSTRING*.

Execute *TSTRING* to calculate the area of the torus with $a = 1.5$ and $b = 1.85$. The labeled answer appears in the status area.

CLEAR
VAR TSTRI
1.5 ▼ 1.85
ENTER



Pausing to Display Output

To pause to display a result:

1. Enter commands to set up the display.
2. Enter the number of seconds you want to pause.
3. Enter the WAIT command (PRG IN menu).

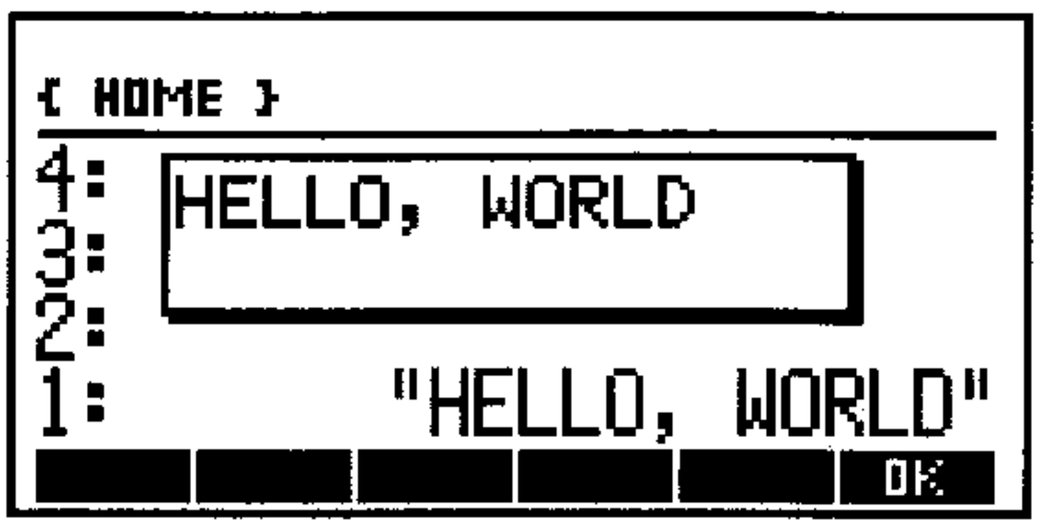
WAIT suspends execution for the number of seconds in level 1. You can use WAIT with DISP to display messages during program execution—for example, to display intermediate program results. (WAIT interprets arguments 0 and −1 differently—see "Using WAIT for Keystroke Input" on page 1-72.)

Using MSGBOX to Display Output

To set up a message box:

1. Enter a message string.
2. Execute the MSGBOX command.

Example: Enter a string "HELLO, WORLD" (ENTER).
Execute MSGBOX (PRG NXT OUT MSGB).
The following message appears:



You must acknowledge a message box by pressing OK or (CANCEL).

Using Menus with Programs

You can use menus with programs for different purposes:

- **Menu-based input.** A program can set up a menu to get input during a halt in a program—then resume executing the same program.
- **Menu-based application.** A program can set up a menu and finish executing, leaving the menu to start executing other related programs.

To set up a built-in or library menu:

1. Enter the menu number.
2. Enter the MENU command (MODES MENU menu).

To set up a custom menu:

1. Enter a list (with { } delimiters) or the name of a list defining the menu actions. If a list of two element lists is given, the first element appears in the menu, but it is the *second* element that is returned to the stack when the menu key is pressed.
2. Activate the menu:
   - To save the menu as the CST menu, enter the MENU command (MODES MENU menu).
   - To make the menu temporary, enter the TMENU command (MODES MENU menu).

The menu isn't displayed until program execution halts.

Menu numbers for built-in menus are listed in chapter 3, under the MENU command. Library menus also have numbers—the library number serves as the menu number. So you can activate applications menus (such as the SOLVE and PLOT menus) and other menus (such as the VAR and CST menus) in programs. The menus behave just as they do during normal keyboard operations.

You create a custom menu to cause the behavior you need in your program—see the topics that follow. You can save the menu as the CST menu, so the user can get it again by pressing CST. Or you can make it *temporary*—it remains active (even after execution stops), but only until a new menu is selected—and it doesn't affect the contents of variable *CST*.

To specify a particular *page* of a menu, enter the number as *m.pp*, where *m* is the menu number and *pp* is the page number (such as 94.02 for page 2 of the TIME menu). If page *pp* doesn't exist, page 1 is displayed (94 gives page 1 of the TIME menu).

Example: Enter 69 MENU to get page 1 of the MODES MISC menu. Enter 69.02 MENU to get page 2 of the MODES MISC menu.

To restore the previous menu:

- Execute 0 MENU.

To recall the menu number for the current menu:

- Execute the RCLMENU command (MODES MENU menu).

Using Menus for Input

To display a menu for input in a program:

1. Set up the menu—see the previous section.
2. Enter a command sequence that halts execution (such as DISP, PROMPT, or HALT).

The program remains halted until it's resumed by a CONT command, such as by pressing ◀ CONT. If you create a custom menu for input, you can include a CONT command to automatically resume the program when you press the menu key.

Example: The following program activates page 1 of the MODES ANGL menu and prompts you to set the angle mode. After you press the menu key, you have to press ◀ CONT to resume execution.

```
« 65 MENU "Select Angle Mode" PROMPT »
```

Example: The *PIE* program on page 2-49 assigns the CONT command to one key in a temporary menu.

Example: The *MNX* program on page 2-22 sets up a temporary menu that includes a program containing CONT to resume execution automatically.

Using Menus to Run Programs

You can use a custom menu to run other programs. That menu can serve as the main interface for an application (a collection of programs).

To create a menu-based application:

1. Create a custom menu list for the application that specifies programs as menu objects.
2. Optional: Create a main program that sets up the application menu—either as the CST menu or as a temporary menu.

Example: The following program, *WGT*, calculates the mass of an object in either English or SI units given the weight. *WGT* displays a temporary custom menu, from which you run the appropriate program. Each program prompts you to enter the weight in the desired unit system, then calculates the mass. The menu remains

active until you select a new menu, so you can do as many calculations as you want.

Enter the following list and store it in LST:

```
{
{ "ENGL" « "ENTER Wt in POUNDS" PROMPT 32.2 / » }
{ "SI" « "ENTER Wt in NEWTONS" PROMPT 9.81 / » }
}
```

' LST STO

Program:

```
« LST TMENU »
```

Comments:
Displays the custom menu stored in *LST*.

ENTER ' WGT STO

Stores the program in *WGT*.

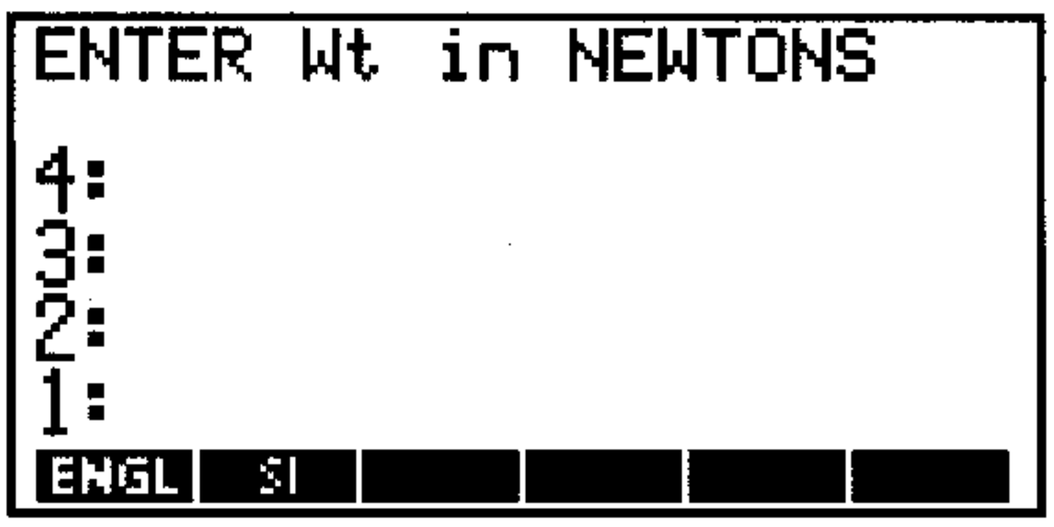
Use *WGT* to calculate the mass of an object of weight 12.5 N. The program sets up the menu, then completes execution.

VAR WGT



Select the SI unit system, which starts the program in the menu list.

SI



Key in the weight, then resume the program.

12.5 ← CONT



Example: The following program, *EIZ*, constructs a custom menu to emulate the HP Solve application for a capacitive electrical circuit. The program uses the equation $E = IZ$, where E is the voltage, I is the current, and Z is the impedance.

Because the voltage, current, and impedance are complex numbers, you can't use the HP Solve application to find solutions. The custom menu in *EIZ* assigns a *direct* solution to the left-shifted menu key for each variable, and assigns *store* and *recall* functions to the unshifted and right-shifted keys—the actions are analogous to the HP Solve

application. The custom menu is automatically stored in *CST*, replacing the previous custom menu—you can press CST to restore the menu.

Program:	**Comments:**
«	
DEG -15 SF -16 SF	Sets Degrees mode. Sets flags −15 and −16 to display complex numbers in polar form. Sets the display mode to 2 Fix.
2 FIX	
{	Starts the custom menu list.
{ "E" { « 'E' STO »	Builds menu key 1 for E.
« I Z * DUP 'E' STO	Unshifted action: stores the
"E: " SWAP + CLLCD	object in *E*. Left-shift action:
1 DISP 1 FREEZE »	calculates $I \times Z$, stores it in *E*,
« E » } }	and displays it with a label. Right-shift action: recalls the object in *E*.
{ "I" { « 'I' STO »	Builds menu key 2.
« E Z / DUP 'I' STO	
"I: " SWAP + CLLCD	
1 DISP 1 FREEZE »	
« I » } }	
{ "Z" { « 'Z' STO »	Builds menu key 3.
« E I / DUP 'Z' STO	
"Z: " SWAP + CLLCD	
1 DISP 1 FREEZE »	
« Z » } }	
}	Ends the list.
MENU	Displays the custom menu.
»	
ENTER ' EIZ STO	Stores the program in *EIZ*.

For a 10-volt power supply at phase angle 0°, you measure a current of 0.37-amp at phase angle 68°. Find the impedance of the circuit using *EIZ*.

← CLEAR VAR EIZ	E I Z

Key in the voltage value.

◀() 10 ▶∠ 0

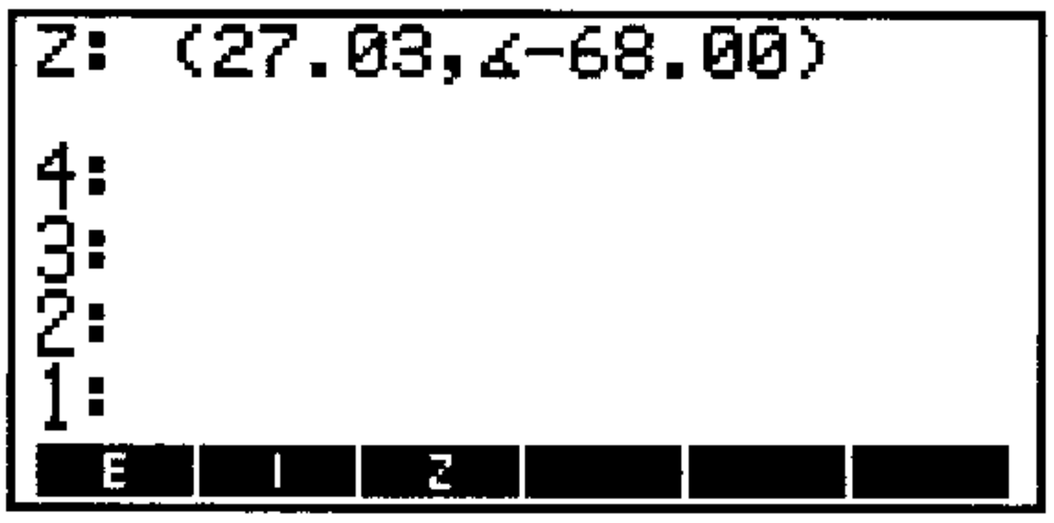


Store the voltage value. Then key in and store the current value.
Solve for the impedance.

E

◀() .37 ▶∠ 68 I

◀ Z



Recall the current and double it. Then find the voltage.

▶ I
2 ×
I
◀ E



Press ◀ MODES FIX STD and NXT MODE ANGL RECT to restore Standard and Rectangular modes.

Turning Off the HP 48 from a Program

To turn off the calculator in a program:

- Execute the OFF command (PRG RUN menu).

The OFF command turns off the HP 48. If a program executes OFF, the program resumes when the calculator is next turned on.

2

Programming Examples

The programs in this chapter demonstrate basic programming concepts. These programs are intended to improve your programming skills, and to provide supplementary functions for your calculator.

At the end of each program, the program's *checksum* and size in bytes are listed to help make sure you typed the program in correctly. (The checksum is a binary integer that uniquely identifies the program based on its contents). To make sure you've keyed the program in correctly, store it in its name, put the name in level 1, then execute the BYTES command (◀ MEMORY BYTES). This returns the program's checksum to level 2, and its size in bytes to level 1. (If you execute BYTES with the program *object* in level 1, you'll get a different byte count.)

The programs in this chapter are also included in the online information of the Program Development Link software for developing HP 48 programs on computers. This software lets you load these programs from the online information into your HP 48 through its serial port.

The examples in this chapter assume the HP 48 is in its initial, default condition—they assume you haven't changed any of the HP 48 operating modes. (To reset the calculator to this condition, see "Memory Reset" in chapter 5 of the *HP 48 User's Guide*.)

Each program listing in this chapter gives the following information:

- A brief description of the program.
- A syntax diagram (where needed) showing the program's required inputs and resulting outputs.
- Discussion of special programming techniques in the program.
- Any other programs needed.
- The program listing.
- The program's checksum and byte size.

Fibonacci Numbers

This section includes three programs that calculate Fibonacci numbers:

- *FIB1* is a user-defined function that is defined *recursively* (that is, its defining procedure contains its own name). *FIB1* is short.
- *FIB2* is a user-defined function with a definite loop. It's longer and more complicated than *FIB1*, but faster.
- *FIBT* calls both *FIB1* and *FIB2* and calculates the execution time of each subprogram.

FIB1 and *FIB2* demonstrate an approach to calculating the *n*th Fibonacci number F_n, where:

$$F_0 = 0,\ F_1 = 1,\ F_n = F_{n-1} + F_{n-2}$$

FIB1 (Fibonacci Numbers, Recursive Version)

Level 1	→	Level 1
n	→	F_n

Techniques used in FIB1

- **IFTE (if-then-else function).** The defining procedure for *FIB1* contains the conditional *function* IFTE, which can take its argument either from the stack or in algebraic syntax.
- **Recursion.** The defining procedure for *FIB1* is written in terms of *FIB1*, just as F_n is defined in terms of F_{n-1} and F_{n-2}.

FIB1 program listing

Program:

```
≪
  → n
  'IFTE(n≤1,
    n,
    FIB1(n-1)+FIB1(n-2))'
≫
```

Comments:

Defines local variable n.
The defining procedure, an algebraic expression. If $n \leq 1$, $F_n = n$, else $F_n = F_{n-1} + F_{n-2}$.

ENTER ' FIB1 STO — Stores the program in *FIB1*.

Checksum: # 41467d (press ' FIB1 ◀ MEMORY BYTES)
Bytes: 113.5

Example: Calculate F_6. Calculate F_{10} using algebraic syntax.

First calculate F_6.

VAR
6 FIB1



Next, calculate F_{10} using algebraic syntax.

' FIB1 ◀ ( ) 10 EVAL



FIB2 (Fibonacci Numbers, Loop Version)

Level 1	→	Level 1
n	→	F_n

Techniques used in FIB2

- **IF ... THEN ... ELSE ... END.** *FIB2* uses the program-structure form of the conditional. (*FIB1* uses IFTE.)

- **START ... NEXT (definite loop).** To calculate F_n, *FIB2* starts with F_0 and F_1 and repeats a loop to calculate successive values of F_i.

FIB2 program listing

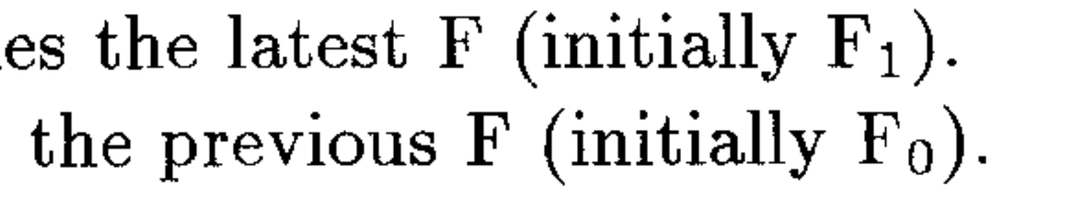
Program:	Comments:
`«`	
`→ n`	Creates a local variable structure.
`«`	
`IF n 1 ≤`	If $n \leq 1$,
`THEN n`	then $F_n = n$;
`ELSE`	otherwise ...
`0 1`	Puts F_0 and F_1 on the stack.
`2 n`	From 2 to n does the following
`START`	loop:
`DUP`	Copies the latest F (initially F_1).
`ROT`	Gets the previous F (initially F_0).
`+`	Calculates the next F (initially F_2).
`NEXT`	Repeats the loop.
`SWAP DROP`	Drops F_{n-1}.
`END`	Ends the ELSE clause.
`»`	Ends the defining procedure.
`»`	
ENTER `'` FIB2 STO	Stores the program in *FIB2*.

Checksum: # 51820d (press `'` FIB2 ← MEMORY BYTES)
Bytes: 89

Example: Calculate F_6 and F_{10}.

Calculate F_6.

VAR
6 FIB2



2-4 Programming Examples

Calculate F_{10}.

10 FIB2



FIBT (Comparing Program-Execution Time)

FIB1 calculates intermediate values F_i more than once, while *FIB2* calculates each intermediate F_i only once. Consequently, *FIB2* is faster. The difference in speed increases with the size of n because the time required for *FIB1* grows exponentially with n, while the time required for *FIB2* grows only linearly with n.

FIBT executes the TICKS command to record the execution time of *FIB1* and *FIB2* for a given value of n.

Level 1	→	Level 3	Level 2	Level 1
n	→	F_n	FIB1 TIME: z	FIB2 TIME: z

Techniques used in FIBT

- **Structured programming.** *FIBT* calls both *FIB1* and *FIB2*.
- **Programmatic use of calculator clock.** *FIBT* executes the TICKS command to record the start and finish of each subprogram.
- **Labeling output.** *FIBT* tags each execution time with a descriptive message.

Required Programs

- *FIB1* (page 2-2) calculates F_n using recursion.
- *FIB2* (page 2-3) calculates F_n using looping.

FIBT program listing

Program:

Comments:

```
«
 DUP TICKS SWAP FIB1
 SWAP TICKS SWAP
 - B→R 8192 /
```

Copies n, then executes *FIB1*, recording the start and stop time. Calculates the elapsed time, converts it to a real number, and converts that number to seconds. Leaves the answer returned by *FIB1* in level 2.

```
 "FIB1 TIME" →TAG
 ROT TICKS SWAP FIB2
 TICKS
 SWAP DROP SWAP
 - B→R 8192 /
 "FIB2 TIME" →TAG
»
```

Tags the execution time. Executes *FIB2*, recording the start and stop time. Drops the answer returned by *FIB2* (*FIB1* returned the same answer). Calculates the elapsed time for *FIB2* and converts to seconds. Tags the execution time.

ENTER ' FIBT STO

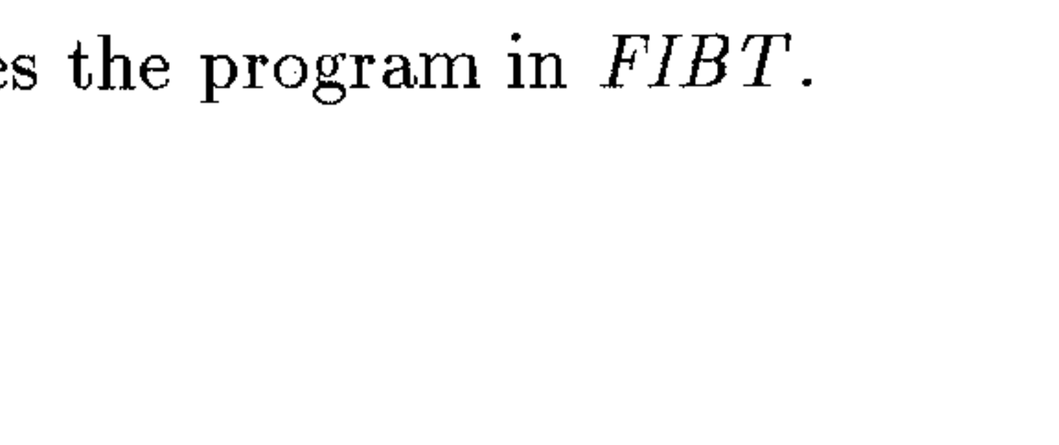
Stores the program in *FIBT*.

Checksum: # 22248d
Bytes: 135

Example: Calculate F_{13} and compare the execution time for the two methods.

Select the VAR menu and do the calculation.

VAR
13 FIBT



F_{13} is 233. *FIB2* takes fewer seconds to execute than *FIB1* (far fewer if n is large). (The times required for the calculations depend on the

contents of memory and other factors, so you may not get the exact times shown above.)

Displaying a Binary Integer

This section contains three programs:

- *PAD* is a utility program that converts an object to a string for right-justified display.
- *PRESERVE* is a utility program for use in programs that change the calculator's status (angle mode, binary base, and so on).
- *BDISP* displays a binary integer in HEX, DEC, OCT, and BIN bases. It calls *PAD* to show the displayed numbers right-justified, and it calls *PRESERVE* to preserve the binary base.

PAD (Pad with Leading Spaces)

PAD converts an object to a string, and if the string contains fewer than 22 characters, adds spaces to the beginning of the string till the string reaches 22 characters.

When a short string is displayed with DISP, it appears *left-justified*: its first character appears at the left end of the display. By adding spaces to the beginning of a short string, *PAD* moves the string to the right. When the string (including leading spaces) reaches 22 characters, it appears *right-justified*: its last character appears at the right end of the display. *PAD* has no effect on longer strings.

Level 1	→	Level 1
object	→	" *object*"

Techniques used in PAD

- **WHILE ... REPEAT ... END (indefinite loop).** The WHILE clause contains a test that executes the REPEAT clause and tests again (if true) or skips the REPEAT clause and exits (if false).

- **String operations.** *PAD* demonstrates how to convert an object to string form, count the number of characters, and combine two strings.

PAD program listing

Program:	**Comments:**
`≪`	
`→STR`	Makes sure the object is in string form. (Strings are unaffected by this command.)
`WHILE`	Repeats if the string contains
`DUP SIZE 22 <`	fewer than 22 characters.
`REPEAT`	Loop-clause adds a leading space.
`" " SWAP +`	
`END`	Ends loop.
`≫`	
ENTER ' PAD STO	Stores the program in *PAD*.

Checksum: # 38912d
Bytes: 61.5

PAD is demonstrated in the program *BDISP*.

PRESERVE (Save and Restore Previous Status)

PRESERVE stores the current calculator (flag) status, executes a program from the stack, and restores the previous status.

Level 1	→	**Level 1**
≪ program ≫	→	*result of program*
'program name'	→	*result of program*

2-8 Programming Examples

Techniques used in PRESERVE

- **Preserving calculator flag status.** *PRESERVE* uses RCLF (*recall flags*) to record the current status of the calculator in a binary integer, and STOF (*store flags*) to restore the status from that binary integer.
- **Local-variable structure.** *PRESERVE* creates a local variable structure to briefly remove the binary integer from the stack. Its defining procedure simply evaluates the program argument, then puts the binary integer back on the stack and executes STOF.
- **Error trapping.** *PRESERVE* uses IFERR to trap faulty program execution on the stack and to restore flags. DOERR shows the error if one occurs.

PRESERVE program listing

Program:	**Comments:**
«	
RCLF	Recalls the list of two 64-bit binary integers representing the status of the 64 system flags and 64 user flags.
→ f	Stores the list in local variable *f*.
«	Begins the defining procedure.
IFERR	Starts the error trap.
EVAL	Executes the program placed on the stack as the level 1 argument.
THEN	If the program caused an error,
f STOF ERRN DOERR	restores flags, shows the error, and aborts execution.
END	Ends the error routine.
f STOF	Puts the list back on the stack, then restores the status of all flags.
»	Ends the defining procedure.
'	
ENTER ' PRESERVE STO	Stores the program in *PRESERVE*.

Checksum: # 7284d
Bytes: 71

PRESERVE is demonstrated in the program *BDISP*.

BDISP (Binary Display)

BDISP displays a real or binary number in HEX, DEC, OCT, and BIN bases.

Level 1	→	Level 1
# *n*	→	# *n*
n	→	*n*

Techniques used in BDISP

- **IFERR ... THEN ... END (error trap).** To accommodate real-number arguments, *BDISP* includes the command R→B (*real-to-binary*). However, this command causes an error if the argument is *already* a binary integer. To maintain execution if an error occurs, the R→B command is placed inside an IFERR clause. No action is required when an error occurs (since a binary number is an acceptable argument), so the THEN clause contains no commands.
- **Enabling LASTARG.** In case an error occurs, the LASTARG recovery feature must be enabled to return the argument (the binary number) to the stack. *BDISP* clears flag −55 to enable this.
- **FOR ... NEXT loop (definite loop with counter).** *BDISP* executes a loop from 1 to 4, each time displaying *n* (the number) in a different base on a different line. The loop counter (named *j* in this program) is a local variable created by the FOR ... NEXT program structure (rather than by a ÷ command), and automatically incremented by NEXT.
- **Unnamed programs as arguments.** A program defined only by its « and » delimiters (not stored in a variable) is not automatically evaluated, but is placed on the stack and can be used as an

argument for a subroutine. *BDISP* demonstrates two uses for unnamed program arguments:

- *BDISP* contains a main program argument and a call to *PRESERVE*. This program argument goes on the stack and is executed by *PRESERVE*.
- *BDISP* also contains four program arguments that "customize" the action of the loop. Each of these contains a command to change the binary base, and each iteration of the loop evaluates one of these arguments.

When *BDISP* creates a local variable for *n*, the defining procedure is an unnamed program. However, since this program is a defining procedure for a local variable structure, it *is* automatically executed.

Required Programs

- *PAD* (page 2-7) expands a string to 22 characters so that DISP shows it right-justified.
- *PRESERVE* (page 2-8) stores the current status, executes the main nested program, and restores the status.

BDISP program listing

Program:	Comments:
«	
«	Begins the main nested program.
DUP	Makes a copy of *n*.
-55 CF	Clears flag −55 to enable LASTARG.
IFERR	Begins error trap.
R→B	Converts *n* to a binary integer.
THEN	If an error occurs, do nothing (no commands in the THEN clause).
END	
→ n	Creates a local variable *n* and
«	begins the defining program.
CLLCD	Clears the display.
« BIN »	Nested program for BIN.
« OCT »	Nested program for OCT.
« DEC »	Nested program for DEC.
« HEX »	Nested program for HEX.

Program:	**Comments:**
1 4	Sets the counter limits.
FOR j	Starts the loop with counter j.
EVAL	Executes one of the nested base programs (initially for HEX).
n →STR	Makes a string showing n in the current base.
PAD	Pads the string to 22 characters.
j DISP	Displays the string in the jth line.
NEXT	Increments j and repeats the loop.
※	Ends the defining program.
3 FREEZE	Freezes the status and stack areas.
※	Ends the main nested program.
PRESERVE	Stores the current flag status, executes the main nested program, and restores the status.
※	
ENTER ' BDISP STO	Stores the program in *BDISP*.

Checksum: # 18055d
Bytes: 191

Example: Switch to DEC base, display #100 in all bases, and check that *BDISP* restored the base to DEC.

Clear the stack and select the MTH BASE menu. Make sure the current base is DEC and enter # 100.

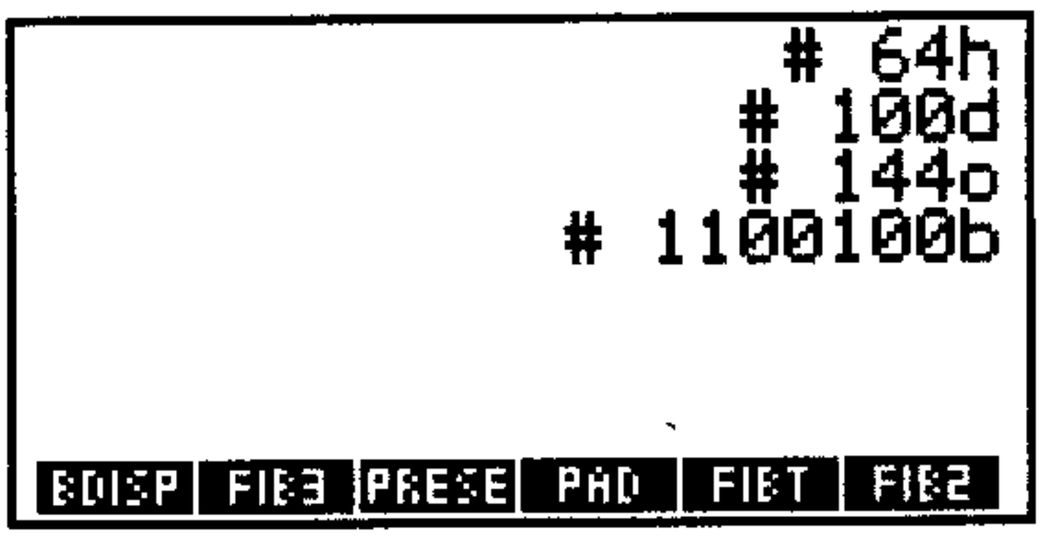
CLEAR
MTH BASE
DEC
100 ENTER



2-12 Programming Examples

Execute *BDISP*.

(VAR) BDISP



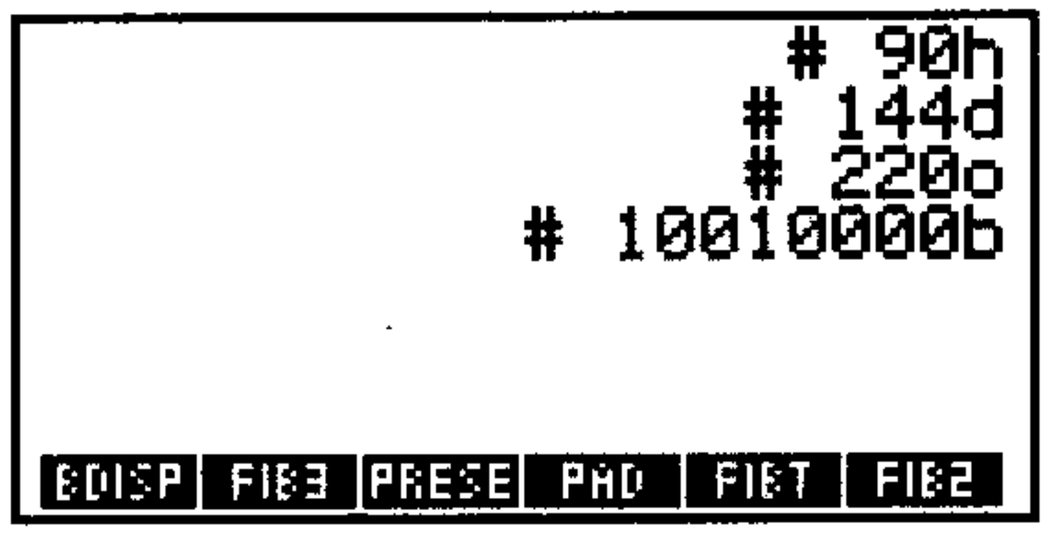
Return to the normal stack display and check the current base.

(CANCEL)
(MTH) BASE



Although the main nested program left the calculator in BIN base, *PRESERVE* restored DEC base. To check that *BDISP* also works for real numbers, try 144.

(VAR)
144 BDISP



Press (CANCEL) to return to the stack display.

Median of Statistics Data

This section contains two programs:

- *%TILE* returns the value of a specified percentile of a list.
- *MEDIAN* uses *%TILE* to calculate the median of the current statistics data.

(*%TILE* and *MEDIAN* are included in the TEACH function's EXAMPLES directory. See the entry for TEACH in chapter 3.)

%TILE (Percentile of a List)

%TILE sorts a list, then returns the value of a specified percentile of the list. For example, typing { *list* } 50 and pressing %TILE returns the median (50th percentile) of the list.

Level 2	Level 1	→	Level 1
{ *list* }	n	→	n^{th} *percentile of sorted list*

Techniques used in %TILE

- **FLOOR and CEIL.** For an integer, FLOOR and CEIL both return that integer; for a noninteger, FLOOR and CEIL return successive integers that bracket the noninteger.
- **SORT.** The SORT command sorts the list elements into ascending order.

%TILE program listing

Program:

Comments:

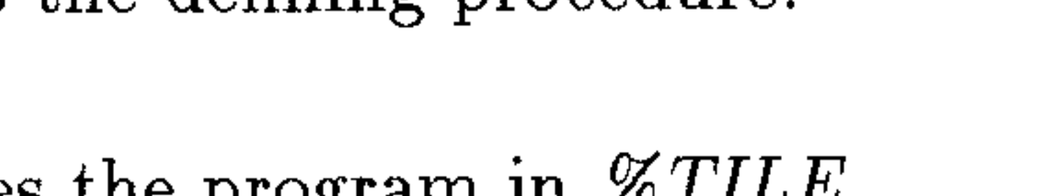
```
«
 SWAP SORT                    Brings the list to level 1 and sorts
                              it.
 DUP SIZE                     Copies the list, then finds its size.
 1 + ROT 100 / *              Calculates the position of the
                              specified percentile.
 → p                          Stores the center position in local
                              variable p.
 «                            Begins the defining procedure.
  DUP                         Makes a copy of the list.
  p FLOOR GET                 Gets the number at or below the
                              center position.
  SWAP                        Moves the list to level 1.
  p CEIL GET                  Gets the number at or above the
                              center position.
  + 2 /                       Calculates the average of the two
                              numbers.
 »                            Ends the defining procedure.
»
```

ENTER ' %TILE STO Stores the program in *%TILE*.

Checksum: # 42718d
Bytes: 99

Example: Calculate the median of the list { 8 3 1 5 2 }.

← {} 8 3 1 5 2 ENTER
VAR 50 %TILE



MEDIAN (Median of Statistics Data)

MEDIAN returns a vector containing the medians of the columns of the statistics data. Note that for a sorted list with an odd number of elements, the median is the value of the center element; for a list with an even number of elements, the median is the average value of the elements just above and below the center.

Level 1	→	Level 1
	→	[x_1 x_2 ... x_m]

Techniques used in MEDIAN

- **Arrays, lists, and stack elements.** *MEDIAN* extracts a column of data from *ΣDAT* in vector form. To convert the vector to a list, *MEDIAN* puts the vector elements on the stack and combines them into a list. From this list the median is calculated using *%TILE*.

  The median for the *m*th column is calculated first, and the median for the first column is calculated last. As each median is calculated, ROLLD is used to move it to the top of the stack.

  After all medians are calculated and positioned on the stack, they're combined into a vector.

- **FOR ... NEXT (definite loop with counter).** *MEDIAN* uses a loop to calculate the median of each column. Because the medians are calculated in reverse order (last column first), the counter is used to reverse the order of the medians.

Required Program

- *%TILE* (page) sorts a list and returns the value of a specified percentile.

MEDIAN program listing

Program:	**Comments:**
«	
RCLΣ	Puts a copy of the current statistics matrix ΣDAT on the stack.
DUP SIZE	Puts the list { n m } on the stack, where n is the number of rows in ΣDAT and m is the number of columns.
OBJ→ DROP	Puts n and m on the stack, and drops the list size.
→ s n m	Creates local variables for s, n, and m.
«	Begins the defining procedure.
'ΣDAT' TRN	Recalls and transposes ΣDAT. Now n is the number of columns in ΣDAT and m is the number of rows. (To key in the Σ character, press →Σ, then delete the parentheses.)
1 m	Specifies the first and last rows.
FOR j	For each row, does the following:
Σ−	Extracts the last row in ΣDAT. Initially this is the mth row, which corresponds to the mth column in the original ΣDAT. (To key in the Σ− command, press ←STAT DATA Σ−.)
OBJ→ DROP	Puts the row elements on the stack. Drops the index list { n }.
n →LIST	Makes an n-element list.
50 %TILE	Sorts the list and calculates its median.
j ROLLD	Moves the median to the proper stack level.
NEXT	Increments j and repeats the loop.

Program:

```
    m  →ARRY

    s  STOΣ

  »
»
```

Comments:

Combines all the medians into an m-element vector.
Restores ΣDAT to its previous value.
Ends the defining procedure.

ENTER ' MEDIAN STO Stores the program in *MEDIAN*.

Checksum: # 57504d
Bytes: 140

Example: Calculate the median of the following data.

$$\begin{bmatrix} 18 & 12 \\ 4 & 7 \\ 3 & 2 \\ 11 & 1 \\ 31 & 48 \\ 20 & 17 \end{bmatrix}$$

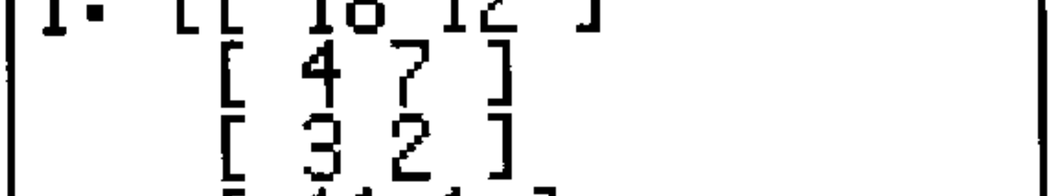
There are two columns of data, so *MEDIAN* will return a two-element vector.

Enter the matrix.

→ MATRIX
18 ENTER 12 ENTER ▼
4 ENTER 7 ENTER
3 ENTER 2 ENTER
11 ENTER 1 ENTER
31 ENTER 48 ENTER
20 ENTER 17 ENTER
ENTER

Store the matrix in ΣDAT, and calculate the median.

← STAT DATA
CLΣ Σ+
VAR MEDIAN



Expanding and Collecting Completely

This section contains two programs:

- *MULTI* repeats a program until the program has no effect on its argument.
- *EXCO* calls *MULTI* to completely expand and collect an algebraic.

MULTI (Multiple Execution)

Given an object and a program that acts on the object, *MULTI* applies the program to the object repeatedly until the program no longer changes the object.

Level 2	Level 1	→	Level 1
object	≪ *program* ≫	→	$object_{result}$

Techniques used in MULTI

- **DO ... UNTIL ... END (indefinite loop).** The DO clause contains the steps to be repeated. The UNTIL clause contains the test that repeats both clauses again (if false) or exits (if true).
- **Programs as arguments.** Although programs are commonly named and then executed by calling their names, programs can also be put on the stack and used as arguments to other programs.
- **Evaluation of local variables.** The program argument to be executed repeatedly is stored in a local variable.

  It's convenient to store an object in a local variable when you don't know beforehand how many copies you'll need. An object stored in a local variable is simply put on the stack when the local variable is evaluated. *MULTI* uses the local variable name to put the program argument on the stack and then executes EVAL to execute the program.

MULTI program listing

Program:	Comments:
«	
→ p	Creates a local variable *p* that contains the program from level 1.
«	Begins the defining procedure.
DO	Begins the DO loop clause.
DUP	Makes a copy of the object, now in level 1.
p EVAL	Applies the program to the object, returning its new version.
DUP	Makes a copy of the new object.
ROT	Moves the old version to level 1.
UNTIL	Begins the DO test clause.
SAME	Tests whether the old version and the new version are the same.
END	Ends the DO structure.
»	Ends the defining procedure.
»	
ENTER ' MULTI STO	Stores the program in *MULTI*.

Checksum: # 34314d
Bytes: 56

MULTI is demonstrated in the next programming example.

EXCO (Expand and Collect Completely)

EXCO repeatedly executes EXPAN on an algebraic until the algebraic doesn't change, then repeatedly executes COLCT until the algebraic doesn't change. In some cases the result will be a number.

Expressions with many products of sums or with powers can take many iterations of EXPAN to expand completely, resulting in a long execution time for *EXCO*.

Level 1	→	Level 1
'*algebraic*'	→	'*algebraic*'
'*algebraic*'	→	*z*

Techniques used in EXCO

- **Subroutines.** *EXCO* calls the program *MULTI* twice. It is more efficient to create program *MULTI* and simply call its name twice than write each step in *MULTI* two times.

Required Programs

- *MULTI* (page 2-19) repeatedly executes the programs that *EXCO* provides as arguments.

EXCO program listing

Program:

Comments:

```
«
  « EXPAN »                    Puts a program on the stack as
                               the level 1 argument for MULTI.
                               The program executes the
                               EXPAN command.
  MULTI                        Executes EXPAN until the
                               algebraic object doesn't change.
  « COLCT »                    Puts another program on the
                               stack for MULTI. The program
                               executes the COLCT command.
  MULTI                        Executes COLCT until the
                               algebraic object doesn't change.
»
ENTER ' EXCO STO               Stores the program in EXCO.
```

Checksum: # 48008d
Bytes: 65.5

Example: Expand and collect completely the expression:

$$3x(4y + z) + (8x - 5z)^2$$

Enter the expression.

' 3 × X ×
◀ ( ) 4 × Y + Z ▶ +
◀ ( ) 8 × X − 5 × Z
▶ y^x 2
ENTER



Select the VAR menu and start the program.

VAR EXCO



Minimum and Maximum Array Elements

This section contains two programs that find the minimum or maximum element of an array:

- *MNX* uses a DO ... UNTIL ... END (indefinite) loop.
- *MNX2* uses a FOR ... NEXT (definite) loop.

MNX (Minimum or Maximum Element—Version 1)

MNX finds the minimum or maximum element of an array on the stack.

Level 1	→	Level 2	Level 1
[[*array*]]	→	[[*array*]]	z_{min} *or* z_{max}

Techniques used in MNX

- **DO ... UNTIL ... END (indefinite loop).** The DO clause contains the sort instructions. The UNTIL clause contains the system-flag test that determines whether to repeat the sort instructions.
- **User and system flags for logic control:**
  - *User* flag 10 defines the sort: When flag 10 is set, *MNX* finds the maximum element; when flag 10 is clear, it finds the minimum element. *You* determine the state of flag 10 at the beginning of the program.
  - *System* flag −64, the Index Wrap Indicator flag, determines when to end the sort. While flag −64 is clear, the sort loop continues. When the index invoked by GETI wraps back to the first array element, flag −64 is *automatically* set, and the sort loop ends.
- **Nested conditional.** An IF ... THEN ... END conditional is nested in the DO ... UNTIL ... END conditional, and determines the following:
  - Whether to maintain the current minimum or maximum element, or make the current element the new minimum or maximum.
  - The sense of the comparison of elements (either < or >) based on the status of flag 10.
- **Custom menu.** *MNX* builds a custom menu that lets you choose whether to sort for the minimum or maximum element. Key 1, labeled MAX, sets flag 10. Key 2, labeled MIN, clears flag 10.
- **Logical function.** *MNX* executes XOR (*exclusive OR*) to test the combined state of the relative value of the two elements and the status of flag 10.

MNX program listing

Program:

Comments:

```
«
 CC "MAX"
   « 10 SF CONT » )
   C "MIN"
   « 10 CF CONT » ))

TMENU
"Sort for MAX or MIN?"
PROMPT
1 GETI
DO
 ROT ROT GETI

 4 ROLL DUP2

 IF
  > 10 FS? XOR

 THEN
   SWAP
 END

 DROP

UNTIL
 -64 FS?

END
SWAP DROP 0 MENU

»
```

Defines the option menu. MAX sets flag 10 and continues execution. MIN clears flag 10 and continues execution.

Displays the temporary menu and a prompt message.

Gets the first element of the array.
Begins the DO loop.
Puts the index and the array in levels 1 and 2, then gets the new array element.
Moves the current minimum or maximum array element from level 4 to level 1, then copies both.
Tests the combined state of the relative value of the two elements and the status of flag 10.
If the new element is either less than the current maximum or greater than the current minimum, swaps the new element into level 1.
Drops the other element off the stack.
Begins the DO test-clause.
Tests if flag −64 is set—if the index reached the end of the array.
Ends the DO loop.
Swaps the index to level 1 and drops it. Restores the last menu.

ENTER ' MNX STO

Stores the program in *MNX*.

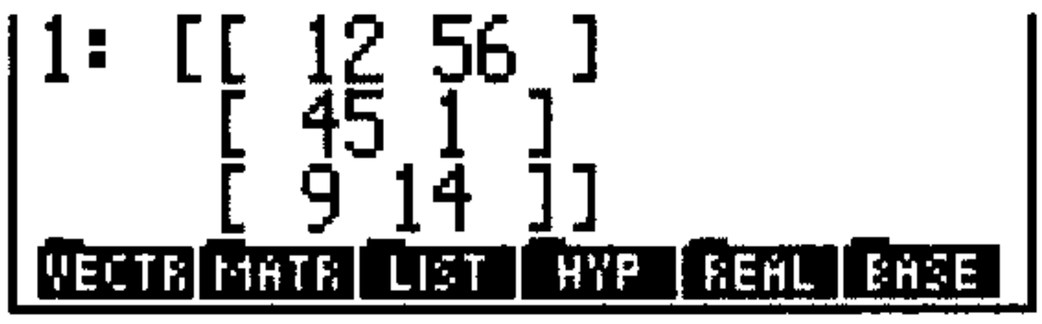
Checksum: # 57179d
Bytes: 210.5

Example: Find the maximum element of the following matrix:

$$\begin{bmatrix} 12 & 56 \\ 45 & 1 \\ 9 & 14 \end{bmatrix}$$

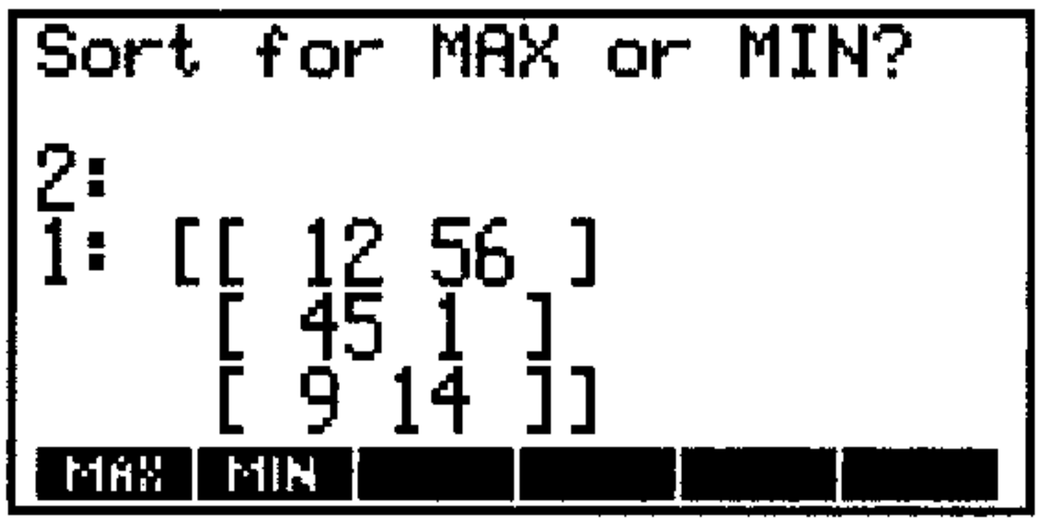
Enter the matrix.

(→) MATRIX
12 ENTER 56 ENTER ▼
45 ENTER 1 ENTER
9 ENTER 14 ENTER
ENTER



Select the VAR menu and execute *MNX*.

VAR MNX



Find the maximum element.

MAX



MNX2 (Minimum or Maximum Element—Version 2)

Given an array on the stack, *MNX2* finds the minimum or maximum element in the array. *MNX2* uses a different approach than *MNX*: it executes OBJ→ to break the array into individual elements on the stack for testing, rather than executing GETI to index through the array.

Level 1	→	Level 2	Level 1
[[*array*]]	→	[[*array*]]	z_{max} *or* z_{min}

Techniques used in MNX2

- **FOR ... NEXT (definite loop).** The initial counter value is 1. The final counter value is $nm - 1$, where nm is the number of elements in the array. The loop-clause contains the sort instructions.
- **User flag for logic control.** *User* flag 10 defines the sort: When flag 10 is set, *MNX2* finds the maximum element; when flag 10 is clear, it finds the minimum element. *You* determine the status of flag 10 at the beginning of the program.
- **Nested conditional.** An IF ... THEN ... END conditional is nested in the FOR ... NEXT loop, and determines the following:
  - Whether to maintain the current minimum or maximum element, or make the current element the new minimum or maximum.
  - The sense of the comparison of elements (either < or >) based on the status of flag 10.
- **Logical function.** *MNX2* executes XOR (*exclusive OR*) to test the combined state of the relative value of the two elements and the status of flag 10.
- **Custom menu.** *MNX2* builds a custom menu that lets you choose whether to sort for the minimum or maximum element. Key 1, labeled MAX, sets flag 10. Key 2, labeled MIN, clears flag 10.

MNX2 program listing

Program:

Comments:

```
«
 << "MAX"
  « 10 SF CONT » >>
  < "MIN"
  « 10 CF CONT » >>
TMENU
"Sort for MAX or MIN?"
 PROMPT
DUP OBJ→
```

Defines the temporary option menu. `MAX` sets flag 10 and continues execution. `MIN` clears flag 10 and continues execution.
Displays the temporary menu and a prompting message.

Copies the array. Returns the individual array elements to levels 2 through $nm+1$, and returns the list containing n and m to level 1.

```
1
SWAP OBJ→
DROP * 1 -
FOR n
 DUP2
IF
 > 10 FS? XOR
THEN
 SWAP
END
```

Sets the initial counter value.
Converts the list to individual elements on the stack.
Drops the list size, then calculates the final counter value $(nm-1)$.
Starts the FOR ... NEXT loop.
Saves the array elements to be tested (initially the last two elements). Uses the last array element as the current minimum or maximum.
Tests the combined state of the relative value of the two elements and the status of flag 10.
If the new element is either less than the current maximum or greater than the current minimum, swaps the new element into level 1.

Program:

```
   DROP

 NEXT

 0 MENU
»
```

Comments:

Drops the other element off the stack.

Ends the FOR ... NEXT loop.

Restores the last menu.

ENTER ' MNX2 STO — Stores the program in *MNX2*.

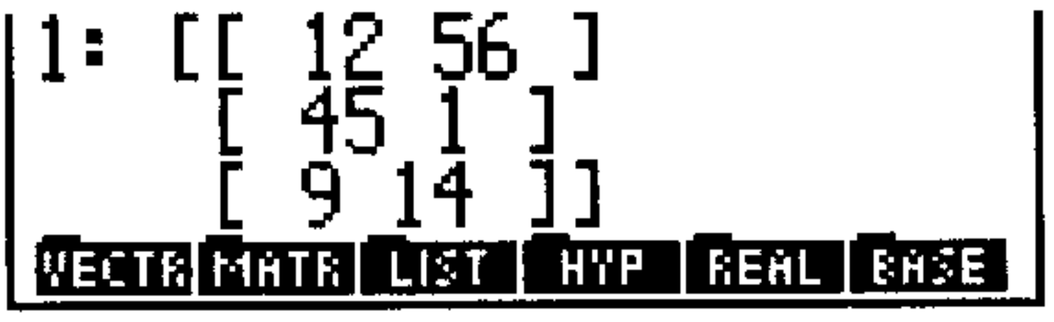
Checksum: # 12277d
Bytes: 200.5

Example: Use *MNX2* to find the minimum element of the matrix from the previous example:

$$\begin{bmatrix} 12 & 56 \\ 45 & 1 \\ 9 & 14 \end{bmatrix}$$

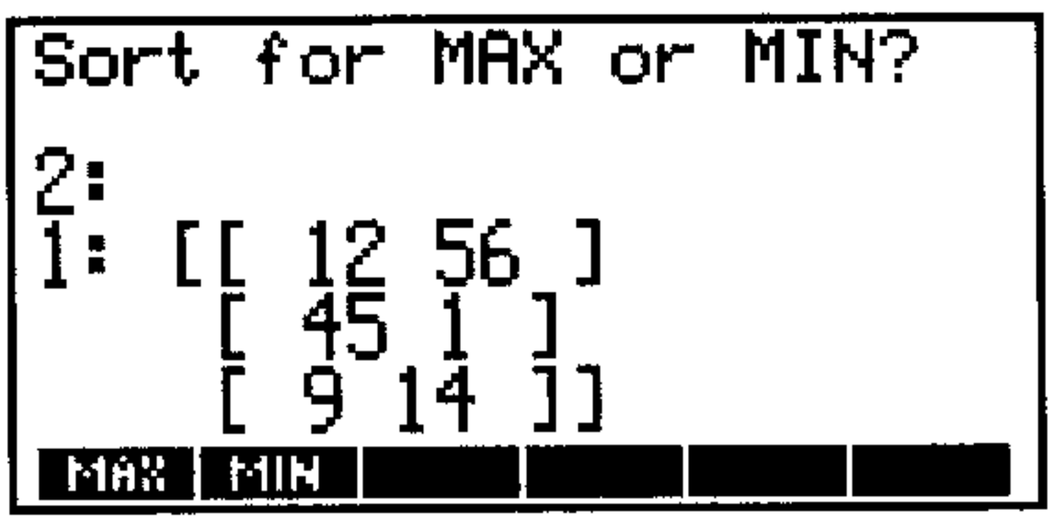
Enter the matrix (or retrieve it from the previous example).

→ MATRIX
12 ENTER 56 ENTER ▼
45 ENTER 1 ENTER
9 ENTER 14 ENTER
ENTER



Select the VAR menu and execute *MNX2*.

VAR MNX2



Find the minimum element.

MIN



Applying a Program to an Array

APLY makes use of list processing to transform each element of an array according to a desired procedure. The input array must be numeric, but the output "array" may be symbolic. Since arrays cannot actually contain symbolic objects, a convention for symbolic "pseudo-arrays" is used. Each row of elements is grouped into a single list and the set of rows is grouped into a list. For example, a 2 × 2 pseudo-array looks like this:

$$\{\ \{\ element_{11}\ \ element_{12}\ \}$$
$$\{\ element_{21}\ \ element_{22}\ \}\ \}$$

The procedure applied to each element must be a program that takes exactly one argument (i.e. the element) and returns exactly one result (i.e. the transformed element).

Level 2	Level 1	→	Level 1
[*array*]	≪ *program* ≫	→	[[*array*]] *or* {{ *array* }}

Techniques used in APLY

- **Manipulating Meta-Objects.** *Meta-objects* are composite objects like arrays and lists that have been disassembled on the stack. APLY illustrates several approaches to manipulating the elements and dimensions of such objects.
- **Application of List Processing.** APLY makes use of DOSUBS (although DOLIST might also have been used) to perform the actual transformation of array elements.
- **Using an IFERR ... THEN ... ELSE ... END Structure.** The entire symbolic pseudo-array case is handled within a error structure—triggered when the →ARRY command generates an error when symbolic elements are present.
- **Using Flags.** User flag 1 is used to track the case when the input array is a vector.

APLY program listing

Program:	Comments:
«	
→ a p	Store the array and program in local variables.
«	Begin the main local variable structure.
1 CF	Make sure the flag 1 is clear to begin the procedure.
a DUP SIZE	Retrieve the dimensions of the array.
DUP SIZE	Determine if the array is a vector.
IF 1 ==	
THEN 1 SF 1 +	If array is a vector,set flag 1 and add a second dimension by treating the vector as an $n \times 1$ matrix.
SWAP OBJ→ OBJ→ DROP	Disassemble the original vector, leaving the element count, n, in level 1.
1 + ROLL	Roll the elements up the stack and bring the "matrix" dimensions of the vector to level 1.
ELSE DROP2 a OBJ→	If array is a matrix, clean up the stack and decompose the matrix into its elements, leaving its dimension list on level 1.
END DUP OBJ→ DROP *	Duplicate the dimension list and compute the total number of elements.
SWAP OVER 2 +	Roll up the element count and combine all elements into a list. Note that the elements in the list are in row-major order.
ROLLD →LIST	
1 p DOSUBS	Recalls the program and uses it as an argument for DOSUBS (DOLIST works in this case as well). Result is a list of transformed elements.
»	

Program:

Comments:

```
OBJ→ 1 + ROLL
```
Disassembles the result list and brings the array dimensions to level 1.

```
IFERR
```
Begins the error-trapping structure. Its purpose is to find and handle the cases when the result list contains symbolic elements.

```
IF 1 FS?
```
Was original array a vector?

```
THEN OBJ→ DROP →LIST
```
If the original array was a vector, then drop the second dimension (1) from the dimension list.

```
END →ARRY
```
Convert the elements into a array with the given dimensions. If there are symbolic elements present, an error will be generated and the error-clause which follows will be executed.

```
THEN
```
Begin the error clause.

```
OBJ→
```
Put the array dimensions on levels 2 and 1. If the array is a vector, level 1 contains a 1.

```
  IF 1 FC?C
```
Is original array a matrix? Clear flag 1 after performing the test.

```
  THEN DROP
```
Drop the number of matrix elements.

```
  END → m n
```
Store the array dimensions in local variables.

```
   « 1 n
    FOR i
```
Begin local variable structure and initiate FOR..NEXT loop for each row.

```
      m →LIST
```
Collect a group of elements into a row (a list).

```
     'm*(n-i)+i'  EVAL
     ROLLD
```
Computes the number of elements to roll so that the next row can be collected.

```
   NEXT
```
Repeat loop for the next row.

```
   n →LIST
```
Gather rows into a list, forming a list of lists (symbolic pseudo-array).

Program:

```
≪
  END 1 CF
≫
```

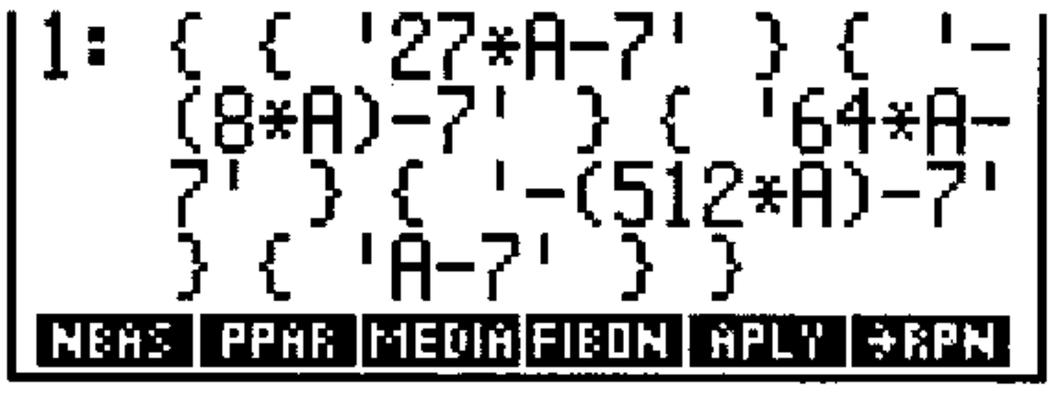
Comments:
Close the local variable structure and end the IFERR..THEN..END structure. Clear flag 1 before exiting the program.

ENTER ' APLY STO Stores the program in *APLY*.

Checksum: # 49768d
Bytes: 319

Example: Apply the function, $f(x) = Ax^3 - 7$ to each element x of the vector [3 -2 4 -8 1].

◀ [] 3 SPC 2 +/- 4 SPC 8 +/- 1 ENTER
◀ « » 3 y^x A × 7 − ENTER
VAR ↑APLY



Converting Between Number Bases

nBASE converts a positive decimal number (x) into a tagged string representation of the equivalent value in a different number base (b). Both x and b must be real numbers. *nBASE* automatically rounds both arguments to the nearest integer.

Level 2	Level 1	→	Level 1
x	*b*	→	*x* base*b*: "*string*"

Techniques used in nBASE

- **String Concatenation and Character Manipulation.** nBASE makes use of several string and character manipulation techniques to build up the result string.
- **Tagged Output.** nBASE labels ("tags") the output string with its original arguments so that the output is a complete record of the command.
- **Indefinite Loops.** nBASE accomplishes most of its work using indefinite loops—both DO..UNTIL..END and WHILE..REPEAT..END loops.

nBASE program listing

Program:	**Comments:**
«	
1 CF 0 RND SWAP 0 RND	Clear flag 1 and round both arguments to integers.
→ b n	Store the base and number in local variables.
«	Begin the outer local variable structure.
n LOG b LOG /	Computes the ratio of the common logarithms of number and base.
10 RND	Rounds result to remove imprecision in last decimal place.
IP n 0	Find the integer part of log ratio, recall the original number, and initialize the counter variable *k* for use in the DO..UNTIL loop.
→ i n k	Store the values in local variables.

Program:

Comments:

```
«
  " "
  DO
```
Begin inner local variable structure, enter an empty string and begin the DO..UNTIL..END loop.

```
    'm' EVAL b i
    'k' EVAL - ^
```
Compute the decimal value of the $(i-k)$th position in the string.

```
  DUP2 MOD
```
Makes a copy of the arguments and computes the decimal value still remaining that must be accounted for by other positions.

```
  IF DUP 0 ==
    'm' EVAL b ≥
    AND
  THEN 1 SF
```
Is the remainder zero *and* $m \geq b$? If the test is true, then set flag 1.

```
  END 'm' STO
  ⁄ IP
```
Store the remainder in m. Compute the number of times the current position-value goes into the remaining decimal value. This is the "digit" that belongs in the current position.

```
  IF DUP 10 ≥
  THEN 55 + CHR
```
Is the "digit" ≥ 10? Then convert the digit into a alphabetic digit (such as A, B, C, ...).

```
  END +
  'k' 1 STO+
```
Append the digit to the current result string and increment the counter variable k.

Program:

```
      UNTIL 'm' EVAL 0 ==

      END
      IF 1 FS?C
      THEN 0 +

   WHILE i 'k' EVAL
    - 0 ≠

   REPEAT 0 +
    1 'k' STO+

   END
     END
    »

  " base" b +
 n SWAP + →TAG

»
```

Comments:

Repeat the DO..UNTIL loop until $m = 0$ (i.e. all decimal value has been accounted for).
Is flag 1 set? Clear the flag after the test.
Then add a placeholder zero to the result string.
Begin WHILE..REPEAT loop to determine if additional placeholder zeros are needed. Loop repeats as long as $i \neq k$.
Add an additional placeholding zero and increment k before repeating the test-clause.
End the WHILE..REPEAT..END loop, the IF..THEN..END structure, and the inner local variable structure.
End the outermost IF..THEN..ELSE..END structure and create the label string and tag the result string using the original arguments.

ENTER ' nBASE STO — Stores the program in *nBASE*.

Checksum: # 36427d
Bytes: 416.5

Example: Convert 1000_{10} to base 23.

1000 ENTER 23 VAR HEX:E

Verifying Program Arguments

The two utility programs in this section verify that the argument to a program is the correct object type.

- *NAMES* verifies that a list argument contains exactly two names.
- *VFY* verifies that the argument is either a name or a list containing exactly two names. It calls *NAMES* if the argument is a list.

You can modify these utilities to verify other object types and object content.

NAMES (Check List for Exactly Two Names)

If the argument for a program is a list (as determined by *VFY*), *NAMES* verifies that the list contains exactly two names. If the list does not contain exactly two names, an error message appears in the status area and program execution is aborted.

Level 1	→	Level 1
{ *valid list* }	→	
{ *invalid list* }	→	(*error message in status area*)

Techniques used in NAMES

- **Nested conditionals.** The outer conditional verifies that there are two objects in the list. If so, the inner conditional verifies that both objects are names.
- **Logical functions.** *NAMES* uses the AND command in the inner conditional to determine if *both* objects are names, and the NOT command to display the error message if they are not both names.

NAMES program listing

Program:	**Comments:**
«	
IF	Starts the outer conditional structure.
OBJ→	Returns the n objects in the list to levels 2 through $(n + 1)$, and returns the list size n to level 1.
DUP 2 SAME	Copies the list size and tests if it's 2.
THEN	If the size is 2, moves the objects to levels 1 and 2, and starts the inner conditional structure.
DROP	
IF	
TYPE 6 SAME	Tests if the first object is a name: returns 1 if so, otherwise 0.
SWAP TYPE 6 SAME	Moves the second object to level 1, then tests if it is a name (returns 1 or 0).
AND	Combines test results: Returns 1 if both tests were true, otherwise returns 0.
NOT	Reverses the final test result.
THEN	If the objects are not both names, displays an error message and aborts execution.
"List needs two names"	
DOERR	
END	Ends the inner conditional structure.
ELSE	If the list size is not 2, drops the list size, displays an error message, and aborts execution.
DROPN	
"Illegal list size"	
DOERR	
END	Ends the outer conditional.
»	
ENTER ' NAMES STO	Stores the program in *NAMES*.

Checksum: # 40666d
Bytes: 141.5

NAMES is demonstrated in the program *VFY*.

VFY (Verify Program Argument)

VFY verifies that an argument on the stack is either a name or a list that contains exactly two names.

Level 1 →	Level 1
'name' →	*'name'*
{ valid list } →	*{ valid list }*
{ invalid list } →	*{ invalid list }* (*and error message in status area*)
invalid object →	*invalid object* (*and error message in status area*)

Techniques used in VFY

- **Utility programs.** *VFY* by itself has little use. However, it can be used with minor modifications by other programs to verify that specific object types are valid arguments.
- **CASE ... END (case structure).** *VFY* uses a case structure to determine if the argument is a list or a name.
- **Structured programming.** If the argument is a list, *VFY* calls *NAMES* to verify that the list contains exactly two names.
- **Local variable structure.** *VFY* stores its argument in a local variable so that it can be passed to *NAMES* if necessary.
- **Logical function.** *VFY* uses NOT to display an error message.

Required Programs

- *NAMES* (page 2-36) verifies that a list argument contains exactly two names.

VFY program listing

Program:

Comments:

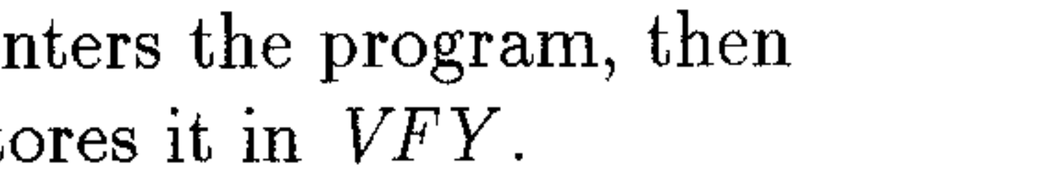
```
«
 DUP                                  Copies the original argument
                                      to leave on the stack.
 DTAG                                 Removes any tags from the
                                      argument for subsequent
                                      testing.
 → argm                               Stores the argument in local
                                      variable argm.
 «                                    Begins the defining procedure.
  CASE                                Begins the case structure.
    argm TYPE 5 SAME                  Tests if the argument is a list.
   THEN                               If so, puts the argument back
    argm NAMES                        on the stack and calls NAMES
   END                                to verify that the list is valid,
                                      then leaves the CASE
                                      structure.
    argm TYPE 6 SAME NOT              Tests if the argument is not a
   THEN                               name. If so, displays an error
    "Not name or list"                message and aborts execution.
    DOERR
   END
 END                                  Ends the CASE structure.
  »                                   Ends the defining procedure.
»
```

ENTER ' VFY STO Enters the program, then stores it in *VFY*.

Checksum: # 36796d
Bytes: 139.5

Example: Execute *VFY* to test the validity of the name argument *BEN*. (The argument is valid and is simply returned to the stack.)

' BEN ENTER
VAR 'VFY'



Example: Execute *VFY* to test the validity of the list argument { *BEN JEFF SARAH* }. Use the name from the previous example, then enter the names *JEFF* and *SARAH* and convert the three names to a list.

' JEFF ENTER
' SARAH ENTER
3 PRG LIST →LIST



Execute *VFY*. Since the list contains too many names, the error message is displayed and execution is aborted.

VAR VFY



Converting Procedures from Algebraic to RPN

This section contains a program, →*RPN*, that converts an algebraic expression into a series (list) of objects in equivalent RPN order. Note that →RPN is a program provided with the TEACH command. You can find it in the EXAMPLES directory by pressing EXAM PRGS →RPN.

Level 1	→	Level 1
'symb'	→	{ *objects* }

Techniques used in →RPN

- **Recursion.** The →RPN program calls itself as a subroutine. This powerful technique works just like calling another subroutine as long as the stack contains the proper arguments before the program calls itself. In this case the level 1 argument is tested first to be sure that it is an algebraic expression before →RPN is called again.

- **Object Type-Checking.** →RPN uses conditional branching that depends on the object type of the level 1 object.
- **Nested Program Structures.** →RPN nests IF ... THEN ... END structures inside FOR ... NEXT loops inside a IF ... THEN ... ELSE ... END structure.
- **List Concatenation.** The result list of objects in RPN order is built by using the ability of the + command to sequentially append additional elements to a list. This is a handy technique for gathering results from a looping procedure.

→RPN program listing

Program:	**Comments:**
«	
OBJ→	Take the expression apart.
IF OVER	If the argument count is
THEN → n f	nonzero, then store the count and the function.
«	Begins local variable defining procedure.
1 n	Begins FOR ... NEXT loop,
FOR i	which converts any algebraic arguments to lists.
IF DUP TYPE 9 SAME	Tests whether argument is an algebraic.
THEN →RPN	If argument is an algebraic, convert it to a list first.
END n ROLLD	Roll down the stack to prepare for the next argument.
NEXT	Repeat the loop for the next argument.
IF DUP TYPE 5 ≠	Tests to see if level 1 object is a list.
THEN 1 →LIST	If not a list, then convert it to one.
END	Ends the IF ... THEN ... END structure.

Program:

```
    IF n 1 >

   THEN 2 n
    START +
    NEXT
  END f +

   ※

 ELSE 1 →LIST SWAP DROP


 END

※
```

Comments:

Tests to see if there is more than one argument.

Combine all of the arguments into a list.

Append the function to the end of the list.

End the local variable defining procedure.

For functions with no arguments, converts to a simple list.

End the IF ... THEN ... ELSE ... END structure.

Checksum: # 28598d
Bytes: 189.5

Example: Convert the following algebraic expression to a series of objects in RPN syntax: '`A*COS(B+√(C/D))-X^3`'.

↑ A × COS B + √ ↶ ( ) C ÷
D ▶ ▶ − X y^x 3 ENTER





Bessel Functions

This section contains a program, *BER*, that calculates the real part $\mathrm{Ber_n(x)}$ of the Bessel function $J_n(xe^{3\pi i/4})$. When $n = 0$,

$$\mathrm{Ber}(x) = 1 - \frac{(x/2)^4}{2!^2} + \frac{(x/2)^8}{4!^2} - \cdots$$

Level 1	→	Level 1
z	→	*Ber*(*z*)

Techniques used in BER

- **Local variable structure.** At its outer level, *BER* consists solely of a local variable structure and so has two properties of a user-defined function: it can take numeric or symbolic arguments from the stack, or it can take arguments in algebraic syntax. However, because *BER* uses a DO ... UNTIL ... END loop, its defining procedure is a *program*. (Loop structures are not allowed in algebraic expressions.) Therefore, unlike user-defined functions, *BER* is not differentiable.

- **DO ... UNTIL ... END loop (indefinite loop with counter).** *BER* calculates successive terms in the series using a counter variable. When the new term does not differ from the previous term to within the 12-digit precision of the calculator, the loop ends.

- **Nested local variable structures.** The outer structure is consistent with the requirements of a user-defined function. The inner structure allows storing and recalling of key parameters.

BER program listing

Program:

Comments:

```
«
 → x
 «
   'x/2' →NUM 2 1
 → xover2 j sum
 «
  DO
    sum
    'sum+(-1)^(j/2)*
    xover2^(2*j)/SQ(j!)'
    EVAL
    2 'j' STO+
    DUP 'sum' STO
  UNTIL
    ==
  END
  sum
 »
»
```

Creates local variable x.
Begins outer defining procedure.
Enters $x/2$, the first counter value, and the first term of the series, then creates local variables.
Begins inner defining procedure.
Begins the loop.
Recalls the old sum and calculates the new sum.
Increments the counter.
Stores the new sum.
Ends the loop clause.
Tests the old and new sums.
Ends the loop.
Recalls the sum.
Ends inner defining procedure.
Ends outer defining procedure.

ENTER ' BER STO — Stores the program in *BER*.

Checksum: # 36388d
Bytes: 200.5

Example: Calculate Ber(3).

VAR
3 BER

```
1:                -.2213802496
 BER  VFY  NAME  MNX2  MNX  EXCO
```

Calculate Ber(2) in algebraic syntax.

' BER ◀ () 2
EVAL

```
1:                 .751734182714
 BER  VFY  NAME  MNX2  MNX  EXCO
```

Animation of Successive Taylor's Polynomials

This section contains three programs that manipulate graphics objects to display a sequence of Taylor's polynomials for the sine function.

- *SINTP* draws a sine curve, and saves the plot in a variable.
- *SETTS* superimposes plots of successive Taylor's polynomials on the sine curve plot from *SINTP*, and saves the resulting graphics objects in a list.
- *TSA* uses the ANIMATE command to display in succession each graphics object from the list built in *SETTS*.

SINTP (Converting a Plot to a Graphics Object)

SINTP draws a sine curve, returns the plot to the stack as a graphics object, and stores that graphics object in a variable. Make sure your calculator is in Radians mode.

Techniques used in SINTP

- **Programmatic use of PLOT commands.** *SINTP* uses PLOT commands to build and display a graphics object.

SINTP program listing

Program:

```
«
  'SIN(X)' STEQ

 FUNCTION '-2*π' →NUM
 DUP NEG XRNG
 -2 2 YRNG
 ERASE DRAW

 PICT RCL 'SINT' STO

»
```

Comments:

Stores the expression for $\sin x$ in *EQ*.

Sets the plot type and x- and y-axis display ranges.

Erases *PICT*, then plots the expression.

Recalls the resultant graphics object and stores it in *SINT*.

ENTER ' SINTP STO — Stores the program in *SINTP*.

Checksum: # 1971d
Bytes: 91.5

SINTP is demonstrated in the program *TSA*.

SETTS (Superimposing Taylor's Polynomials)

SETTS superimposes successive Taylor's polynomials on a sine curve and stores each graphics object in a list.

Techniques used in SETTS

- **Structured programming.** *SETTS* calls *SINTP* to build a sine curve and convert it to a graphics object.
- **FOR ... STEP (definite loop).** *SETTS* calculates successive Taylor's polynomials for the sine function in a definite loop. The loop counter serves as the value of the order of each polynomial.
- **Programmatic use of PLOT commands.** *SETTS* draws a plot of each Taylor's polynomial.
- **Manipulation of graphics objects.** *SETTS* converts each Taylor's polynomial plot into a graphics object. Then it executes + to combine each graphics object with the sine curve stored in *SINT*, creating nine new graphics objects, each the superposition of a

Taylor's polynomial on a sine curve. *SETTS* then puts the nine new graphics objects, and the sine curve graphics object itself, in a list.

SETTS program listing

Program:	Comments:
«	
SINTP	Plots a sine curve and stores the graphics object in *SINT*.
1 17 FOR n	Sets the range for the FOR loop using local variable *n*.
'SIN(X)' 'X' n TAYLR	Plots the Taylor's polynomial of order *n*.
STEQ ERASE DRAW	
PICT RCL SINT +	Returns the plot to the stack as a graphics object and executes + to superimpose the sine plot from *SINT*.
2 STEP	Increments the loop counter *n* by 2 and repeats the loop.
SINT	Puts the sine curve graphics object on the stack, then builds a list containing it and the nine graphics objects created in the loop. Stores the list in *TSL*.
10 →LIST	
'TSL' STO	
»	
ENTER ' SETTS STO	Stores the program in *SETTS*.

Checksum: # 28102d
Bytes: 138.5

SETTS is demonstrated in the program *TSA*.

TSA (Animating Taylor's Polynomials)

TSA displays in succession each graphics object created in *SETTS*.

Techniques used in TSA

- **Passing a global variable.** Because *SETTS* takes several minutes to execute, *TSA* does not call *SETTS*. Instead, you must first execute *SETTS* to create the global variable *TSL* containing the list of

graphics objects. *TSA* simply executes that global variable to put the list on the stack.

- **FOR ... NEXT (definite loop).** *TSA* executes a definite loop to display in succession each graphics object from the list.

TSA program listing

Program:

Comments:

```
«
 TSL OBJ→                          Puts the list TSL on the stack
                                   and converts it to 10 graphics
                                   objects and the list count.
 { { #0 #0 } .5 0 } +              Set up the parameters for
                                   ANIMATE.
 ANIMATE                           Displays the graphics in
                                   succession.
 11 DROPN                          Removes the graphics objects and
                                   list count from the stack.
»
```

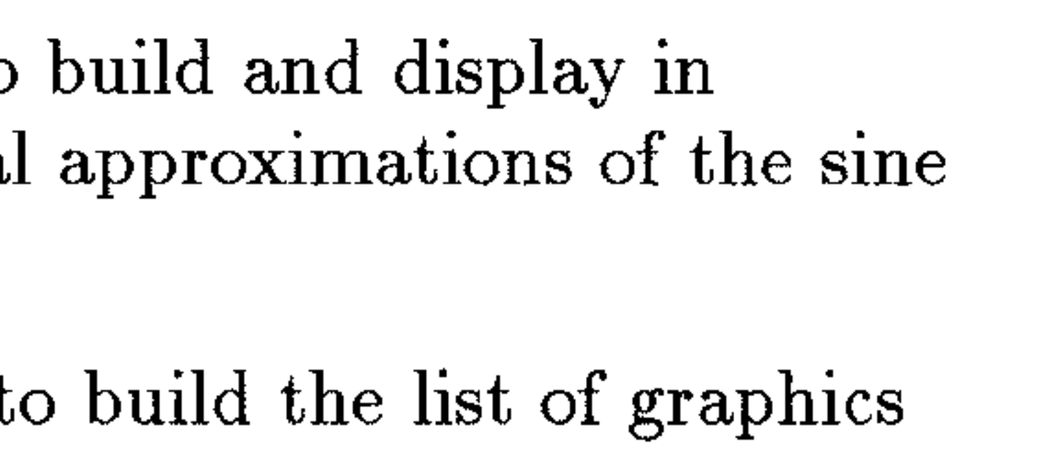
ENTER ' TSA STO — Stores the program in *TSA*.

Checksum: # 59350d
Bytes: 96.5

Example: Execute *SETTS* and *TSA* to build and display in succession a series of Taylor's polynomial approximations of the sine function.

Set Radians mode and execute *SETTS* to build the list of graphics objects. (*SETTS* takes several minutes to execute.) Then execute *TSA* to display each plot in succession. The display shows *TSA* in progress.

← RAD (if necessary)
VAR SETTS
TSA



Press CANCEL to stop the animation. Press ← RAD to restore Degrees mode.

Programmatic Use of Statistics and Plotting

This section describes a program *PIE* you can use to draw pie charts. *PIE* prompts for single variable data, stores that data in the statistics matrix ΣDAT, then draws a labeled pie chart that shows each data point as a percentage of the total.

Techniques used in PIE

- **Programmatic use of PLOT commands.** *PIE* executes XRNG and YRNG to define x- and y-axis display ranges in user units, and executes ARC and LINE to draw the circle and individual slices.
- **Programmatic use of matrices and statistics commands.**
- **Manipulating graphics objects.** *PIE* recalls *PICT* to the stack and executes GOR to merge the label for each slice with the plot.
- **FOR ... NEXT (definite loop).** Each slice is calculated, drawn, and labeled in a definite loop.
- **CASE ... END structure.** To avoid overwriting the circle, each label is offset from the midpoint of the arc of the slice. The offset for each label depends on the position of the slice in the circle. The CASE ... END structure assigns an offset to the label based on the position of the slice.
- **Preserving calculator flag status.** Before specifying Radians mode, *PIE* saves the current flag status in a local variable, then restores that status at the end of the program.
- **Nested local variable structures.** At different parts of the process, intermediate results are saved in local variables for convenient recall as needed.
- **Temporary menu for data input.**

PIE program listing

Program:

Comments:

```
«
 RCLF → flags

 «
RAD
<< "SLICE" Σ+ >>
  < >
  < "CLEAR" CLΣ >
  < > < >
  < "DRAW" CONT >>

TMENU
"Key values into
SLICE,■DRAW
restarts program."
PROMPT
ERASE 1 131 XRNG
1 64 YRNG CLLCD
"Please wait...■
Drawing Pie Chart"
1 DISP
(66,32) 20 0 6.28
ARC
PICT RCL →LCD
RCLΣ TOT /


DUP 100 *

→ prcnts

 «
  2 π →NUM * *
  0
```

Recalls the current flag status and stores it in variable *flags*.

Sets Radians mode.

Defines the input menu: Key 1 executes Σ+ to store each data point in ΣDAT, key 3 clears ΣDAT, and key 6 continues program execution after data entry.

Displays the temporary menu.

Prompts for inputs.

■ represents the newline character (ENTER) after you enter the program on the stack.

Erases the current *PICT* and sets plot parameters.

Displays "drawing" message.

Draws the circle.

Displays the empty circle.

Recalls the statistics data matrix, computes totals, and calculates the proportions.

Converts the proportions to percentages.

Stores the percentage matrix in *prcnts*.

Multiplies the proportion matrix by 2π, and enters the initial angle (0).

Program:

```
→ prop angle

«
 prop SIZE OBJ→
 DROP SWAP
 FOR n
 (66,32) prop n GET
 'angle' STO+

 angle COS angle SIN
 R→C 20 * OVER +
 LINE
 PICT RCL
 angle prop n GET
 2 / - DUP DUP
 COS SWAP SIN R→C
 26 * (66,32) +
 SWAP
 CASE
   DUP 1.5 ≤
  THEN
   DROP
  END
   DUP 4.4 ≤
  THEN
   DROP 15 -
  END
   5 <
  THEN
   (3,2) +
  END
 END
»
```

Comments:

Stores the angle matrix in *prop* and angle in *angle*.

Sets up 1 to *m* as loop counter range.

Begins loop-clause.

Puts the center of the circle on the stack, then gets the *n*th value from the proportion matrix and adds it to *angle*.

Computes the endpoint and draws the line for the *n*th slice.

Recalls *PICT* to the stack.

For labeling the slice, computes the midpoint of the arc of the slice.

Starts the CASE structure to test *angle* and determine the offset value for the label.
From 0 to 1.5 radians, doesn't offset the label.

From 1.5 to 4.4 radians, offsets the label 15 user units left.

From 4.4 to 5 radians, offsets the label 3 units right and 2 units up.

Ends the CASE structure.

Program:

```
    prcnts n GET
    1 RND
    →STR "%" +

    1 →GROB
      GOR DUP PICT STO
      →LCD
    NEXT
    ( ) PVIEW
  ≫
≫
flags STOF
  ≫ 0 MENU
≫
```

Comments:

Gets the nth value from the percentage matrix, rounds it to one decimal place, and converts it to a string with "%" appended.

Converts the string to a graphics object.

Adds the label to the plot and stores the new plot.

Displays the updated plot.

Ends the loop structure.

Displays the finished plot.

Restores the original flag status.

Restores the previous menu. (You must first press CANCEL to clear the plot.)

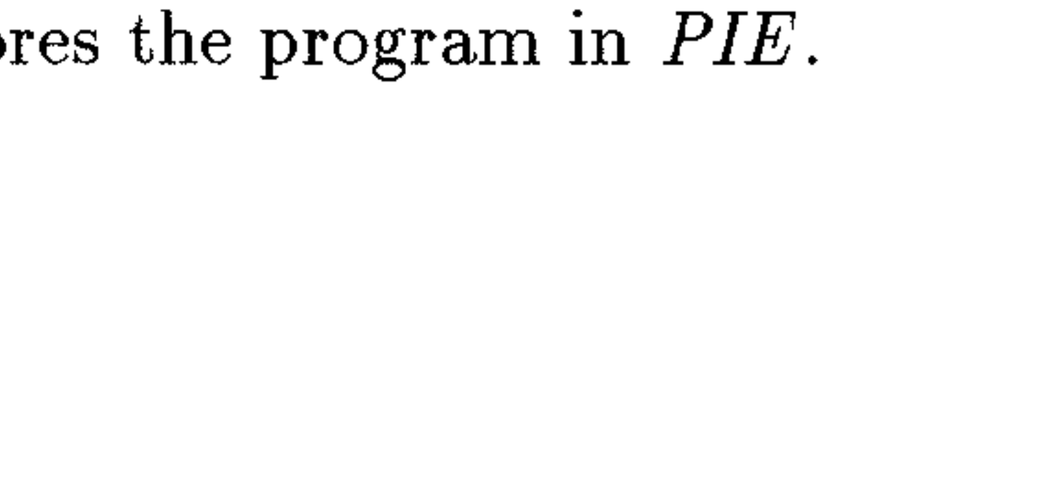
ENTER ' PIE STO

Stores the program in *PIE*.

Checksum: # 1177d
Bytes: 765

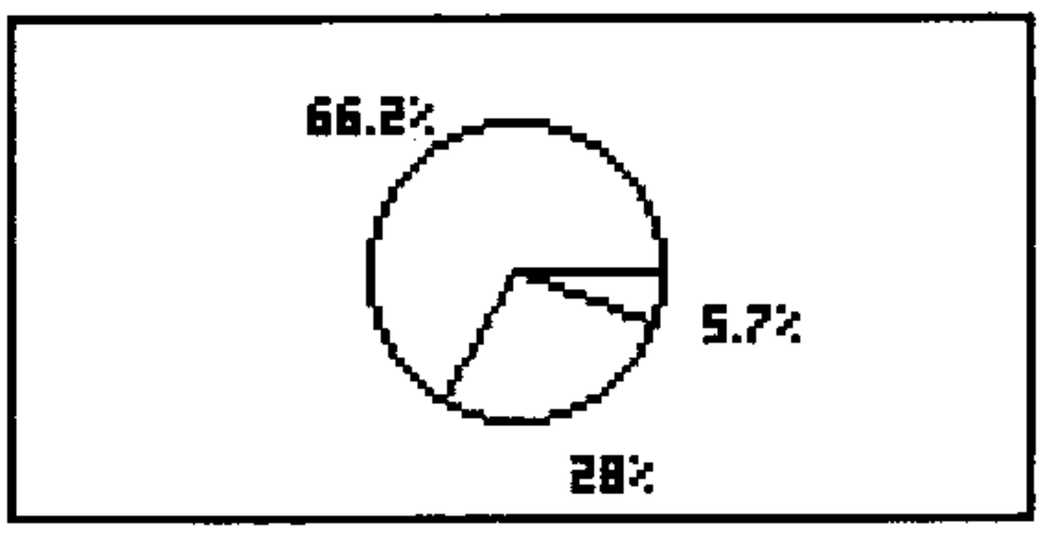
Example: The inventory at Fruit of the Vroom, a drive-in fruit stand, includes 983 oranges, 416 apples, and 85 bananas. Draw a pie chart to show each fruit's percentage of total inventory.

VAR PIE



Clear the current statistics data. (The prompt is removed from the display.) Key in the new data and draw the pie chart.

```
CLEAR
983 SLICE
416 SLICE
85 SLICE
DRAW
```



Press CANCEL to return to the stack display.

Trace Mode

This section contains two programs, $\alpha ENTER$ and $\beta ENTER$, which together provide "trace mode" for the HP 48 using an external printer. To turn on "trace mode," set flag −63 and activate User mode. To turn off "trace mode," clear flag −63 or turn off User mode.

Techniques used in αENTER and βENTER

- **Vectored ENTER.** Setting flag −63 and activating User mode turns on vectored ENTER. When vectored ENTER is turned on and variable $\alpha ENTER$ exists, the command-line text is put on the stack as a string and $\alpha ENTER$ is evaluated. Then, if variable $\beta ENTER$ exists, the command that triggered the command-line processing is put on the stack as a string and $\beta ENTER$ is evaluated.

αENTER program listing

Program:	Comments:
«	
PR1	Prints the command line text,
OBJ→	then converts the string to an object and evaluates it.
»	
ENTER ' αENTER STO	Stores the program in $\alpha ENTER$. (Press α → A to type α. You *must* use this name.)

Checksum: # 51789d
Bytes: 25.5

βENTER program listing

Program:	Comments:
≪	
PR1 DROP	Prints the command that caused the processing, then drops it and prints the stack in compact form.
PRSTC	
≫	
ENTER ' βENTER STO	Stores the program in *βENTER*. (Press α → B to type β. You *must* use this name.)

Checksum: # 37631d
Bytes: 28

Inverse-Function Solver

This section describes the program *ROOTR*, which finds the value of x at which $f(x) = y$. You supply the variable name for the program that calculates $f(x)$, the value of y, and a guess for x (in case there are multiple solutions).

Level 3	Level 2	Level 1	→	Level 1
'*function name*'	*y*	x_{guess}	→	*x*

Techniques used in ROOTR

- **Programmatic use of root-finder.** *ROOTR* executes ROOT to find the desired x-value.
- **Programs as arguments.** Although programs are commonly named and then executed by calling their names, programs can also be put on the stack and used as arguments to other programs.

ROOTR program listing

Program:

```
«
 → fname yvalue xguess
 «
  xguess 'XTEMP' STO
  « XTEMP fname
  yvalue - »
  'XTEMP'
  xguess
  ROOT
 »
 'XTEMP' PURGE
»
```

Comments:

Creates local variables.
Begins the defining procedure.
Creates variable $XTEMP$ (to be solved for).
Enters program that evaluates $f(x) - y$.
Enters name of unknown variable.
Enters guess for $XTEMP$.
Solves program for $XTEMP$.
Ends the defining procedure.
Purges the temporary variable.

(ENTER) (') ROOTR (STO) Stores the program in $ROOTR$.

Checksum: # 13007d
Bytes: 163

Example: Assume you often work with the expression $3.7x^3 + 4.5x^2 + 3.9x + 5$ and have created the program $X \rightarrow FX$ to calculate the value:

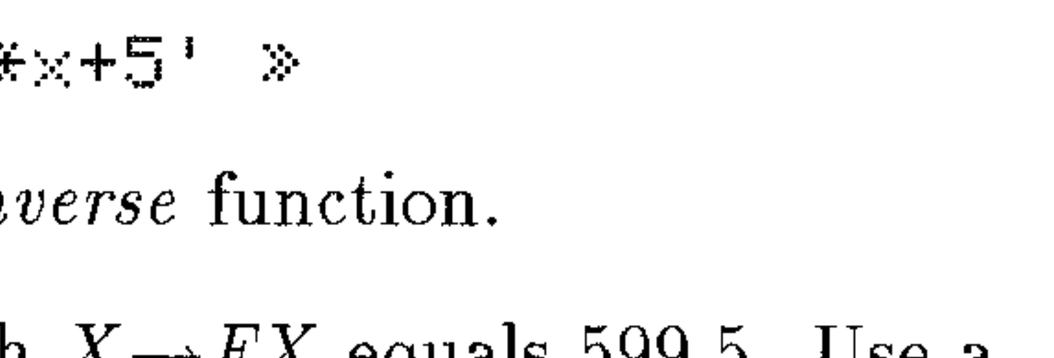
```
« → x '3.7*x^3+4.5*x^2+3.9*x+5' »
```

You can use $ROOTR$ to calculate the *inverse* function.

Example: Find the value of x for which $X \rightarrow FX$ equals 599.5. Use a guess in the vicinity of 1.

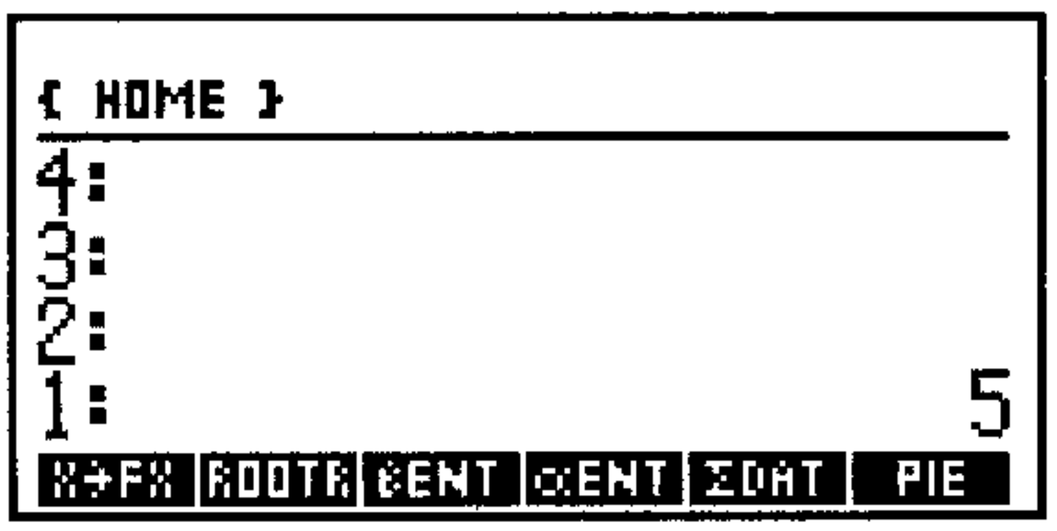
Start by keying in $X \rightarrow FX$:

(◀) («») (▶) (→) x (SPC) (') 3.7 (×) x (y^x) 3 (+) 4.5 (×) x (y^x) 2 (+) 3.9 (×) x (+) 5 (ENTER)



Store the program in $X \rightarrow FX$, then enter the program name, the y-value 599.5, and the guess 1, and execute *ROOTR*:

' X→FX STO
' VAR X→FX ENTER
599.5 ENTER 1 ROOTR



Animating a Graphical Image

Program *WALK* shows a small person walking across the display. It animates this custom graphical image by incrementing the image position in a loop structure.

Techniques used in WALK

- **Custom graphical image.** (Note that the programmer compiles the full information content of the graphical image before writing the program by building the image *interactively* in the Graphics environment and then returning it to the command line.)
- **FOR ... STEP (definite loop).** *WALK* uses this loop to animate the graphical image. The ending value for the loop is MAXR. Since the counter value cannot exceed MAXR, the loop executes indefinitely.

WALK program listing

Program:

Comments:

```
«
 GROB 9 15 E300
 14001500 1C001400 E300
 8000C110 AA009400 9000
 41002200 14102800
```

Puts the graphical image of the walker in the command line. (Note that the hexadecimal portion of the graphics object is a continuous integer E300 ... 2800. The linebreaks do *not* represent spaces.)

```
→ walk
```

Creates local variable *walk* containing the graphics object.

```
«
 ERASE { # 0d # 0d }
 PVIEW
 { # 0d # 25d }
 PICT OVER walk GXOR

 5 MAXR FOR i

  i 131 MOD R→B

  # 25d 2 →LIST

   PICT OVER walk GXOR

   PICT ROT walk GXOR

  5 STEP

»
```

Clears *PICT*, then displays it.

Puts the first position on the stack and turns on the first image. This readies the stack and *PICT* for the loop.

Starts the loop to generate horizontal coordinates indefinitely.

Computes the horizontal coordinate for the next image.

Specifies a fixed vertical coordinate. Puts the two coordinates in a list.

Displays the new image, leaving its coordinates on the stack.

Turns off the old image, removing its coordinates from the stack.

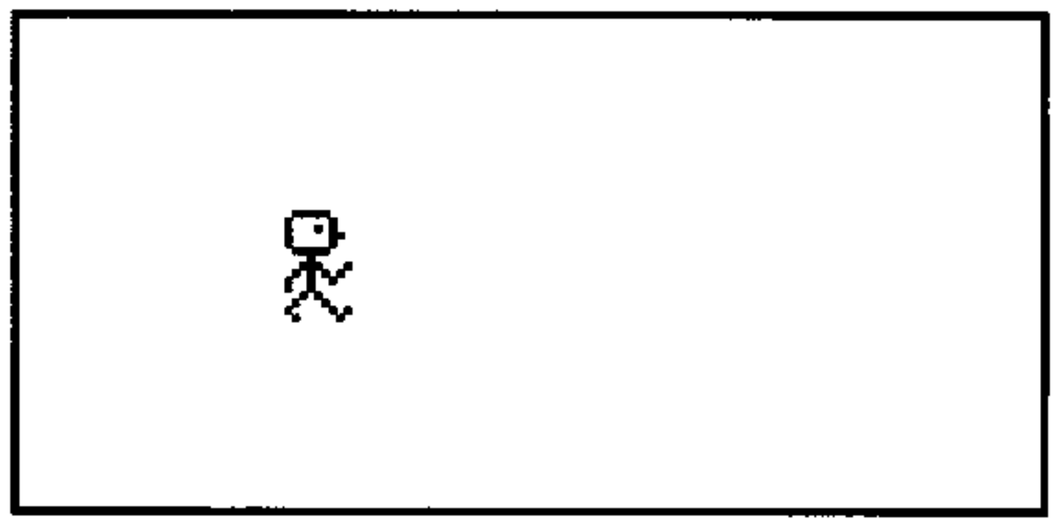
Increments the horizontal coordinate by 5.

ENTER ' WALK STO — Stores the program in *WALK*.

Checksum: # 18146d
Bytes: 240.5

Example: Send the small person out for a walk.

VAR WALK



Press CANCEL when you think the walker's tired.

3

Command Reference

This chapter contains an alphabetical listing of the programmable commands and functions available on the HP 48. The listings include the following information:

- a brief definition of what the command or function does
- a stack diagram showing the arguments it requires (if any)
- the keys to press to gain access to it
- any flags that may affect how it works
- additional information about how it works and how to use it
- an example of its use
- related commands or functions

The next few pages explain how to read the stack diagrams in the command reference, how commands are alphabetized, and the meaning of the command classifications at the upper right corner of each stack diagram.

How to Read Stack Diagrams

Each entry in the command reference includes a *stack diagram*. This is a table showing the *arguments* that the command, function, or analytic function takes from the stack and the *results* that it returns to the stack. The "→" character in the table separates the arguments from the results. The stack diagram for a command may contain more than one "argument → result" line, reflecting all possible combinations of arguments and results for that command.

Consider this example:

ACOS

Arc Cosine Analytic Function: Returns the value of the angle having the given cosine.

{ }

Level 1	→	Level 1
z	→	arc cos *z*
'*symb*'	→	'ACOS(*symb*)'

This diagram indicates that the *analytic function* ACOS (*Arc Cosine*) takes a single argument from level 1 and returns one result (to level 1). ACOS can take either a real or complex number or an algebraic object as its argument. In the first case, it returns the numeric arccosine; in the second, it returns the symbolic arccosine expression of the argument.

Some commands affect a calculator state—a mode, a reserved variable, a flag, or a display— without taking any arguments from the stack or returning any results to the stack. No stack diagrams are shown for these commands.

Parallel Processing with Lists

Commands that can use the parallel list processing feature are denoted by the "{ }" symbol located above the stack diagram. This feature is discussed in greater detail in Appendix G.

As a rule-of-thumb, a command can use parallel list processing if all the following are true:

- The command checks for valid argument types. Commands that apply to all object types, such as DUP, SWAP, ROT, and so forth, do not use parallel list processing.
- The command takes exactly one, two, three, four, or five arguments, none of which may itself be a list. Commands, such as →LIST,

that have an indefinite number of arguments do not use parallel list processing.
- The command isn't a programming branch command (IF, FOR, CASE, NEXT, and so forth).

The HP 48 also has a few commands (PURGE and DELKEYS are examples) that have list processing capability built into their definitions, and so do not also use the parallel list processing feature.

How Commands Are Alphabetized

Commands appear in alphabetical order. Command names that contain special (non-alphabetic) characters are organized as follows:

- For commands that contain *both* special and alphabetic characters:
  - A special character at the *start* of a command name is *ignored*. Therefore, the command ∗H follows the command GXOR and precedes the command HALT.
  - A special character *within* or at the *end* of a command name is considered to follow "Z" at the end of the alphabet. Therefore, the command R→B follows the command RSD and precedes the command R→C.
- Commands that contain *only* special characters appear at the end of the dictionary.

Classification of Operations

The command dictionary contains HP 48 *commands*, *functions*, and *analytic functions*. Commands are calculator operations that can be executed from a program. Functions are commands that can be included in algebraic objects. Analytic functions are functions for which the HP 48 provides an inverse and a derivative. There are also four non-programmmable *operations* (DBUG, NEXT, SST, and SST↓) that are included with the programmable commands as a convenience because they are used interactively while programming.

The definitions of the abbreviations used for argument and result objects are contained in the following table, "Terms Used in Stack Diagrams." Often, descriptive subscripts are added to convey more information.

Terms Used in Stack Diagrams

Term	Description
arg	Argument.
[*array*]	Real or complex vector or matrix.
[*C-array*]	Complex vector or matrix.
date	Date in form MM.DDYYYY or DD.MMYYYY.
{ *dim* }	List of one or two array dimensions (real numbers).
' *global* '	Global name.
grob	Graphics object.
HMS	A real-number time or angle in hours-minutes-seconds format.
{ *list* }	List of objects.
local	Local name.
[[*matrix*]]	Real or complex matrix.
n or m	Positive integer real number (rounded if noninteger).
$:n_{\text{port}}:name_{\text{backup}}$	Backup identifier.
$:n_{\text{port}}:n_{\text{library}}$	Library identifier.
#n	Binary integer.
{ #n #m }	Pixel coordinates. (Uses binary integers.)
' *name* '	Global or local name.
obj	Any object.
PICT	Current graphics object.
≪ *program* ≫	Program.
[*R-array*]	Real vector or matrix.
"*string*"	Character string.
' *symb* '	Expression, equation, or name treated as an algebraic.
T/F	Test result used as an *argument*: zero (false) or non-zero (true)real number.
0/1	Test result *returned* by a command: zero (false) or one (true).
time	Time in form HH.MMSSs.
[*vector*]	Real or complex vector.
x or y	Real number.
x_unit	Unit object, or a real number treated as a dimensionless object.
(x , y)	Complex number in rectangular form, or user-unit coordinate.
z	Real or complex number.

3-4 Command Reference

ABS

Absolute Value Function: Returns the absolute value of its argument.

{}

Level 1	→	Level 1
x	→	|*x*|
(*x*,*y*)	→	$\sqrt{x^2+y^2}$
x_unit	→	|*x*|_*unit*
[*array*]	→	|| *array* ||
'*symb*'	→	'ABS(*symb*)'

Keyboard Access:

MTH REAL NXT ABS

MTH MATR NORM ABS

MTH NXT CMPL ABS

MTH VECTR ABS

Affected by Flags: Numerical Results (−3)

Remarks: ABS has a derivative (SIGN) but not an inverse.

In the case of an array, ABS returns the Frobenius (Euclidean) norm of the array, defined as the square root of the sum of the squares of the absolute values of all n elements. That is,

$$\sqrt{\sum_{i=1}^{n} |z_i|^2}$$

Related Commands: NEG, SIGN

ACK

Acknowledge Alarm Command: Acknowledges the oldest past-due alarm.

Keyboard Access: ↩ (TIME) ALRM ACK

Affected by Flags: Repeat Alarms Not Rescheduled (−43), Acknowledged Alarms Saved (−44)

Remarks: ACK clears the alert annunciator if there are both no other past-due alarms and no other active alert sources (such as a low battery condition).

ACK has no effect on control alarms. Control alarms that come due are automatically acknowledged *and* saved in the system alarm list.

Related Commands: ACKALL

ACKALL

Acknowledge All Alarms Command: Acknowledges all past-due alarms.

Keyboard Access: ↩ (TIME) ALRM ACKA

Affected by Flags: Repeat Alarms Not Rescheduled (−43), Acknowledged Alarms Saved (−44)

Remarks: ACKALL clears the alert annunciator if there are no other active alert sources (such as a low battery condition).

ACKALL has no effect on control alarms. Control alarms that come due are automatically acknowledged *and* saved in the system alarm list.

Related Commands: ACK

ACOS

Arc Cosine Analytic Function: Returns the value of the angle having the given cosine.

{ }

Level 1	→	Level 1
z	→	arc cos *z*
'*symb*'	→	'ACOS(*symb*)'

Keyboard Access: ← ACOS

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Angle Mode (−17, −18)

Remarks: For a real argument x in the domain $-1 \leq x \leq 1$, the result ranges from 0 to 180 degrees (0 to π radians; 0 to 200 grads).

A real argument outside of this domain is converted to a complex argument $z = x + 0i$, and the result is complex.

The inverse of COS is a *relation*, not a function, since COS sends more than one argument to the same result. The inverse relation for COS is expressed by ISOL as the *general solution*

```
'≡1*ACOS(Z)+2*π*ℤ1'
```

The function ACOS is the inverse of a *part* of COS, a part defined by restricting the domain of COS such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of COS are called the *principal values* of the inverse relation. ACOS in its entirety is called the *principal branch* of the inverse relation, and the points sent by ACOS to the boundary of the restricted domain of COS form the *branch cuts* of ACOS.

The principal branch used by the HP 48 for ACOS was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued arc cosine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

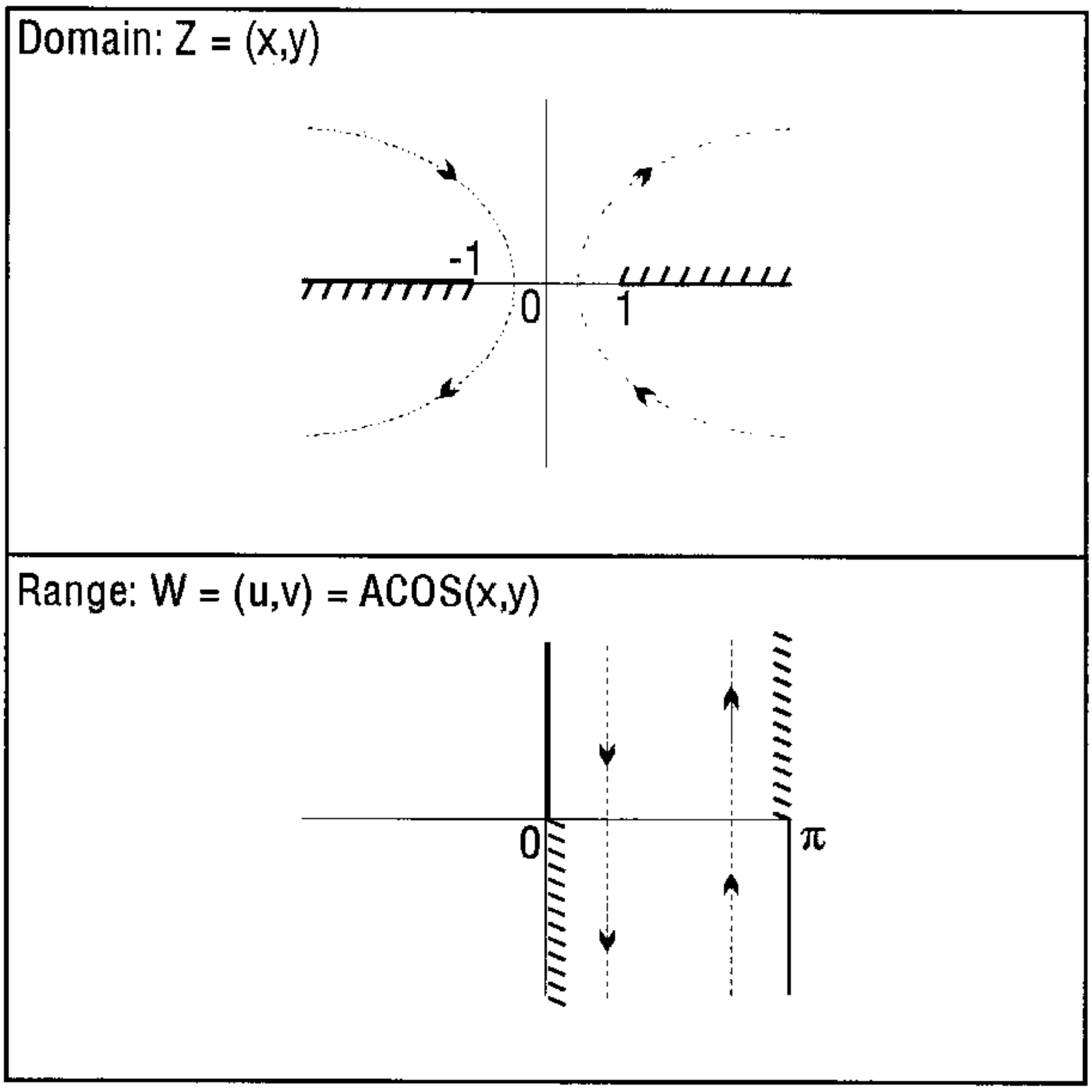
ACOS

The graphs below show the domain and range of ACOS. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

These graphs show the inverse relation 'S1*ACOS(Z)+2*π*N1' for the case *s1*=1 and *n1*=0. For other values of *s1* and *n1*, the vertical band in the lower graph is translated to the right or to the left. Taken together, the bands cover the whole complex plane, which is the domain of COS.

View these graphs with domain and range reversed to see how the domain of COS is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (*x*, *y*). COS sends this domain onto the whole complex plane in the range W = (*u*, *v*) = COS(*x*, *y*) in the upper graph.

Related Commands: ASIN, ATAN, COS, ISOL



Branch Cuts for ACOS(Z)

ACOSH

Inverse Hyperbolic Cosine Analytic Function: Returns the inverse hyperbolic cosine of the argument.

{}

Level 1	→	Level 1
z	→	acosh *z*
'*symb*'	→	'ACOSH(*symb*)'

Keyboard Access: (MTH) HYP ACOSH

Affected by Flags: Principal Solution (−1), Numerical Results (−3)

Remarks: For real arguments $|x| < 1$, ACOSH returns the complex result obtained for the argument $(x, 0)$.

The inverse of ACOSH is a *relation*, not a function, since COSH sends more than one argument to the same result. The inverse relation for COSH is expressed by ISOL as the *general solution*

```
'±1*ACOSH(Z)+2*π*i*n1'
```

The function ACOSH is the inverse of a *part* of COSH, a part defined by restricting the domain of COSH such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of COSH are called the *principal values* of the inverse relation. ACOSH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ACOSH to the boundary of the restricted domain of COSH form the *branch cuts* of ACOSH.

The principal branch used by the HP 48 for ACOSH was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued hyperbolic arc cosine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

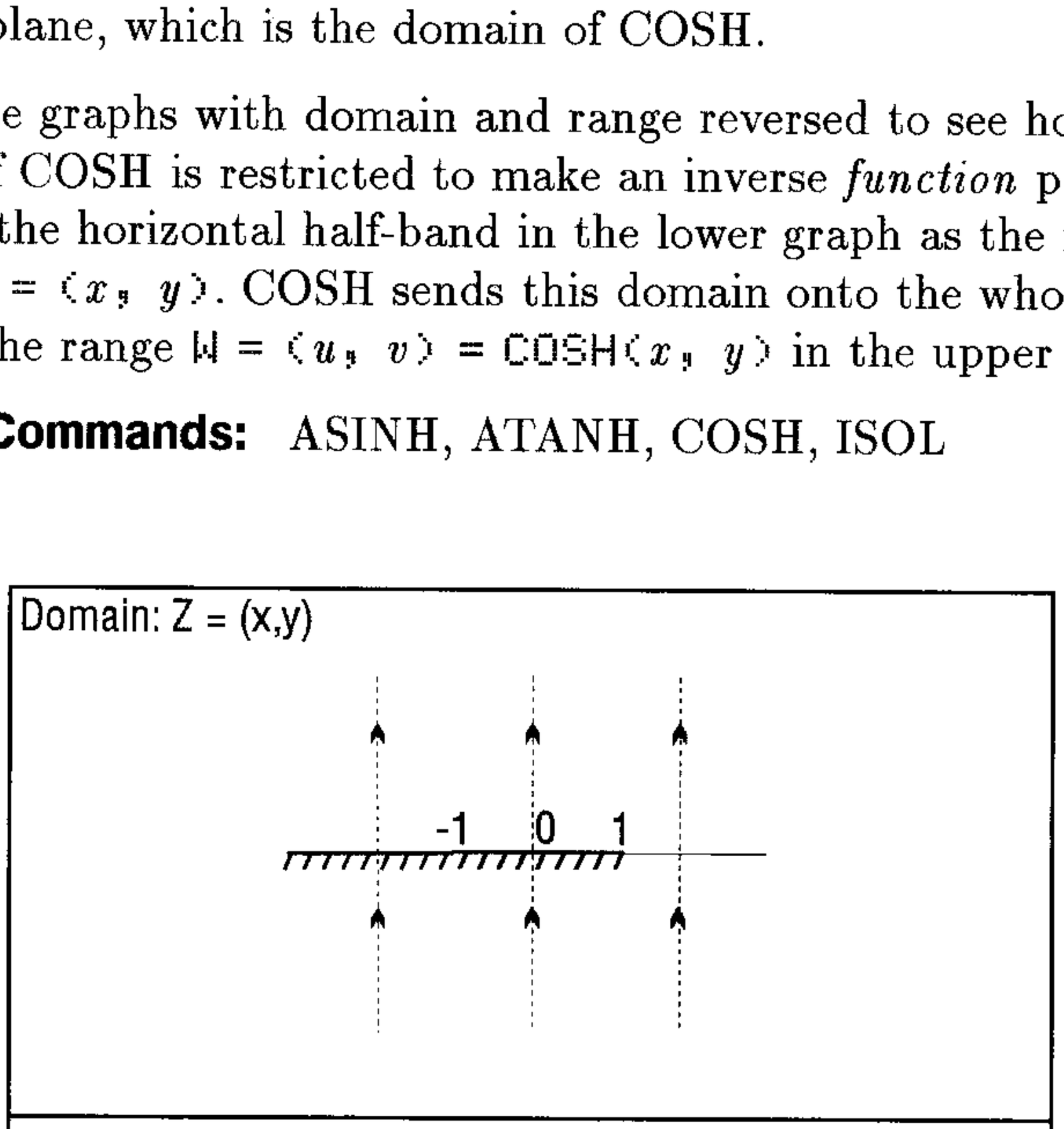
The graphs below show the domain and range of ACOSH. The graph of the domain shows where the branch cut occurs: the heavy solid line

marks one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

These graphs show the inverse relation '≡1*ACOSH(Z)+2*π*i*⌐1' for the case *s1*=1 and *n1*=0. For other values of *s1* and *n1*, the horizontal half-band in the lower graph is rotated to the left and translated up and down. Taken together, the bands cover the whole complex plane, which is the domain of COSH.

View these graphs with domain and range reversed to see how the domain of COSH is restricted to make an inverse *function* possible. Consider the horizontal half-band in the lower graph as the restricted domain Z = (*x*, *y*). COSH sends this domain onto the whole complex plane in the range W = (*u*, *v*) = COSH(*x*, *y*) in the upper graph.

Related Commands: ASINH, ATANH, COSH, ISOL



Branch Cut for ACOSH(Z)

ADD

Add List Command: Adds corresponding elements of two lists or adds a number to each of the elements of a list.

{ }

Level 2	Level 1	→	Level 1
{ $list_1$ }	{ $list_2$ }	→	{ $list_{result}$ }
{ $list$ }	$obj_{non-list}$	→	{ $list_{result}$ }
$obj_{non-list}$	{ $list$ }	→	{ $list_{result}$ }

Keyboard Access: (MTH) LIST ADD

Affected by Flags: None

Remarks: ADD executes the + command once for each of the elements in the list. If two lists are the arguments, they must have the same number of elements as ADD will execute the + command once for each corresponding pair of elements. If one argument is a non-list object, ADD will attempt to execute the + command using the non-list object and each element of the list argument, returning the result to the corresponding position in the result. (See the + command entry to see the object combinations that are defined.) If an undefined addition is encountered, a Bad Argument Type error results.

Related Commands: ΔLIST, ΠLIST, ΣLIST

ALOG

Common Antilogarithm Analytic Function: Returns the common antilogarithm; that is, 10 raised to the given power.

{ }

Level 1	→	Level 1
z	→	10^z
'*symb*'	→	'ALOG(*symb*)'

Keyboard Access: ◀ 10^x

Affected by Flags: Numerical Results (−3)

Remarks: For complex arguments:

$$10^{(x,y)} = e^{cx} \cos cy + i\ e^{cx} \sin cy$$

where $c = \ln 10$.

Related Commands: EXP, LN, LOG

AMORT

Amortize Command: Amortizes a loan or investment based upon the current amortization settings.

{ }

Level 1	→	Level 3	Level 2	Level 1
n	→	*principal*	*interest*	*balance*

Keyboard Access: ◀ SOLVE TVM AMORT

Affected by Flags: Financial Payment Mode (−14)

Remarks: Values must be stored in the TVM variables ($I\%YR$, PV, PMT, and PYR). The number of payments n is taken from level 1 and flag −14.

Related Commands: TVM, TVMBEG, TVMEND, TVMROOT

AND

And Function: Returns the logical AND of two arguments.

{ }

Level 2	Level 1	→	Level 1
$\#n_1$	$\#n_2$	→	$\#n_3$
"$string_1$"	"$string_2$"	→	"$string_3$"
T/F_1	T/F_2	→	0/1
T/F	'$symb$'	→	'T/F AND $symb$'
'$symb$'	T/F	→	'$symb$ AND T/F'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ AND $symb_2$'

Keyboard Access:

MTH BASE NXT LOGIC AND

PRG TEST NXT AND

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: When the arguments are binary integers or strings, AND does a bit-by-bit (base 2) logical comparison.

- An argument that is a binary integer is treated as a sequence of bits as long as the current wordsize. Each bit in the result is determined by comparing the corresponding bits (bit_1 and bit_2) in the two arguments as shown in the following table:

AND

bit_1	bit_2	bit_1 **AND** bit_2
0	0	0
0	1	0
1	0	0
1	1	1

- An argument that is a string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must have the same number of characters.

When the arguments are real numbers or symbolics, AND simply does a true/false test. The result is 1 (true) if both arguments are non-zero; it is 0 (false) if either or both arguments are zero. This test is usually done to compare two test results.

If either or both of the arguments are algebraic expressions, then the result is an algebraic of the form '$symb_1$ AND $symb_2$'. Execute →NUM (or set flag −3 before executing AND) to produce a numeric result from the algebraic result.

Related Commands: NOT, OR, XOR

ANIMATE

Animate Command: Displays graphic objects in sequence.

Level n+1...Level 2	Level 1	→	Level 1
$grob_n \ldots grob_1$	n_{grobs}	→	*same stack*
$grob_n \ldots grob_1$	{ *n* { #X #Y } *delay rep* }	→	*same stack*

Keyboard Access:

PRG GROB NXT ANIM

Affected by Flags: None

Remarks: ANIMATE displays a series of graphics objects (or variables containing them) one after the other. You can use a list to specify the area of the screen you want to animate (pixel coordinates $\#X$ and $\#Y$), the number of seconds before the next grob is displayed (*delay*), and the number of times the sequence is run (*rep*). If *rep* is set to 0, the sequence is played one million times, or until you press CANCEL.

If you use a list on level 1, all parameters must be present.

The animation displays PICT while displaying the grobs. The grobs and the animate parameters are left on the stack.

Example: The following program draws half a cylinder and rotates it:

```
« PARSURFACE { 'COS(X)' 'SIN(X)' Y }
  STEQ
    « I 180 I + XXRNG ERASE DRAW PICT RCL
    »
  I 0 359 8 SEQ OBJ→ ANIMATE DROPN
»
```

This program also illustrates the use of SEQ and PARSURFACE. You can adjust the increment value used with SEQ (8 is used here) to change the number of images drawn by the program, or to use less memory.

APPLY

Apply to Arguments Function: Creates an expression from the specified function name and arguments.

Level 2	Level 1	→	Level 1
{ $symb_1$ ... $symb_n$ }	'*name*'	→	'*name*($symb_1$ ... $symb_n$)'

Keyboard Access: ◀ SYMBOLIC NXT APPLY

APPLY

Affected by Flags: None

Remarks: A user-defined function f that checks its arguments for special cases often can't determine whether a symbolic argument x represents one of the special cases. The function f can use APPLY to create a new expression '$f(x)$'. If the user now evaluates '$f(x)$', x is evaluated before f, so the argument to f will be the result obtained by evaluating x.

The algebraic syntax for APPLY is this:

'APPLY(*name*, $symb_1$, ... , $symb_n$)'

When evaluated in an algebraic expression, APPLY evaluates the arguments (to resolve local names in user-defined functions) before creating the new object.

Example: The following user-defined function *Asin* is a variant of the built-in function ASIN. *Asin* checks for special numerical arguments. If the argument on the stack is symbolic (the second case in the case structure), *Asin* uses APPLY to return the expression 'Asin(*argument*)'.

```
«
 → argument
 «
  CASE
    -3 FS? THEN argument ASIN END
    { 6 7 9 } argument TYPE POS
     THEN 'APPLY(Asin,argument)' EVAL END
    'argument==1' THEN 'π/2' END
    'argument==-1' THEN '-π/2' END
   argument ASIN
  END
 »
»
```

ENTER ' Asin STO

Related Commands: QUOTE, |

ARC

Draw Arc Command: Draws an arc in *PICT* counterclockwise from $x_{\theta 1}$ to $x_{\theta 2}$, with its center at the coordinate specified in level 4 and its radius specified in level 3.

Level 4	Level 3	Level 2	Level 1	→	Level 1
$(x,\ y)$	x_{radius}	$x_{\theta 1}$	$x_{\theta 2}$	→	
$\{\#n\ \#m\}$	$\#n_{\text{radius}}$	$x_{\theta 1}$	$x_{\theta 2}$	→	

Keyboard Access: PRG PICT ARC

Affected by Flags: Angle Mode (−17 and −18)

The setting of flags −17 and −18 determine the interpretation of $x_{\theta 1}$ and $x_{\theta 2}$ (degrees, radians, or grads).

Remarks: ARC always draws an arc of constant radius in pixels, even when the radius and center are specified in user-units, regardless of the relative scales in user-units of the x- and y-axes. With user-unit arguments, the arc starts at the pixel specified by $(x,\ y) + (a,\ b)$, where $(a,\ b)$ is the rectangular conversion of the polar coordinate $(x_{\text{radius}},\ x_{\theta 1})$. The resultant distance in pixels from the starting point to the center pixel is used as the actual radius, r'. The arc stops at the pixel specified by $(r',\ x_{\theta 2})$.

If $x_{\theta 1} = x_{\theta 2}$, ARC plots one point. If $|x_{\theta 1} - x_{\theta 2}| > 360$ degrees, 2π radians, or 400 grads, ARC draws a complete circle.

Example: In Degrees mode, with the x-axis display range (XRNG) specified as −6.5 to 6.5, the command sequence `(0,0) 1 0 90 ARC` draws an arc counterclockwise from 0 to 90 degrees with a constant radius of 10 pixels.

Related Commands: BOX, LINE, TLINE

ARCHIVE

Archive HOME Command: Creates a backup copy of the *HOME* directory (that is, all variables), the user-key assignments, and the alarm catalog in the specified backup object (: n_{port} : *name*) in independent RAM.

Level 1	→	Level 1
:n_{port} :*name*	→	
:IO :*name*	→	

Keyboard Access: ◀ MEMORY NXT ARCHI

Affected by Flags: I/O Device (−33), I/O Messages (−39) *if* the argument is : IO: *name*

Remarks: The specified port number can be 0 through 33. The port used (except 0) must be configured as independent RAM. (See FREE.) An error will result if there is not enough independent RAM in the specified port to copy the HOME directory.

If the backup object is : IO: *name*, then the copied directory is transmitted in binary via Kermit protocol through the current I/O port to the specified filename.

To save flag settings, execute RCLF and store the resulting list in a variable.

Related Commands: RESTORE

ARG

Argument Function: Returns the (real) polar angle θ of a complex number (x, y).

{ }

Level 1	→	Level 1
(x,y)	→	θ
'*symb*'	→	'ARG(*symb*)'

Keyboard Access: MTH NXT CMPL ARG

Affected by Flags: Angle mode (−17, −18)

Remarks: The polar angle θ is equal to:

- arc tan y/x for $x \geq 0$
- arc tan $y/x + \pi$ sign y for $x < 0$, Radians mode
- arc tan $y/x + 180$ sign y for $x < 0$, Degrees mode
- arc tan $y/x + 200$ sign y for $x < 0$, Grads mode

A real argument x is treated as the complex argument $(x, 0)$.

ARRY→

Array to Stack Command: Takes an array and returns its elements as separate real or complex numbers. Also returns a list of the dimensions of the array.

Level 1	→	Level nm+1 ... Level 2	Level 1
[*vector*]	→	$z_1 \ \ldots \ z_n$	{ $n_{element}$ }
[[*matrix*]]	→	$z_{11} \ \ldots \ z_{nm}$	{ n_{row} m_{col} }

ARRY→

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: The command OBJ→ includes this functionality. ARRY→ is included for compatibility with the HP 28S. ARRY→ is not in a menu.

If the argument is an n-element vector, the first element is returned to level $n + 1$ (not level $nm + 1$), and the nth element to level 2.

Related Commands: →ARRY, DTAG, EQ→, LIST→, OBJ→, STR→

→ARRY

Stack to Array Command: Returns a vector of n real or complex elements or a matrix of $n \times m$ real or complex elements.

Level nm+1 ... Level 2	Level 1	→	Level 1
$z_1 \ \cdots \ z_n$	$n_{element}$	→	[*vector*]
$z_{11} \ \cdots \ z_{nm}$	{ n_{row} m_{col} }	→	[[*matrix*]]

Keyboard Access: PRG TYPE →ARR

Affected by Flags: None

Remarks: The elements of the result array should be entered into the stack in row order, with z_{11} (or z_1) in level $nm + 1$ (or $n + 1$), and z_{nm} (or z_n) in level 2. If one or more of the elements is a complex number, the result array will be complex.

Related Commands: ARRY→, LIST→, →LIST, OBJ→, STR→, →TAG, →UNIT

ASIN

Arc Sine Analytic Function: Returns the value of the angle having the given sine.

{ }

Level 1	→	Level 1
z	→	arc sin *z*
'*symb*'	→	'ASIN(*symb*)'

Keyboard Access: ← ASIN

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Angle Mode (−17, −18)

Remarks: For a real argument x in the domain $-1 \leq x \leq 1$, the result ranges from −90 to +90 degrees ($-\pi/2$ to $+\pi/2$ radians; −100 to +100 grads).

A real argument outside of this domain is converted to a complex argument $z = x + 0i$, and the result is complex.

The inverse of SIN is a *relation*, not a function, since SIN sends more than one argument to the same result. The inverse relation for SIN is expressed by ISOL as the *general solution*

```
'ASIN(Z)*(-1)^n1+π*n1'
```

The function ASIN is the inverse of a *part* of SIN, a part defined by restricting the domain of SIN such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of SIN are called the *principal values* of the inverse relation. ASIN in its entirety is called the *principal branch* of the inverse relation, and the points sent by ASIN to the boundary of the restricted domain of SIN form the *branch cuts* of ASIN.

The principal branch used by the HP 48 for ASIN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued arc sine function occurs where the corresponding real-valued function

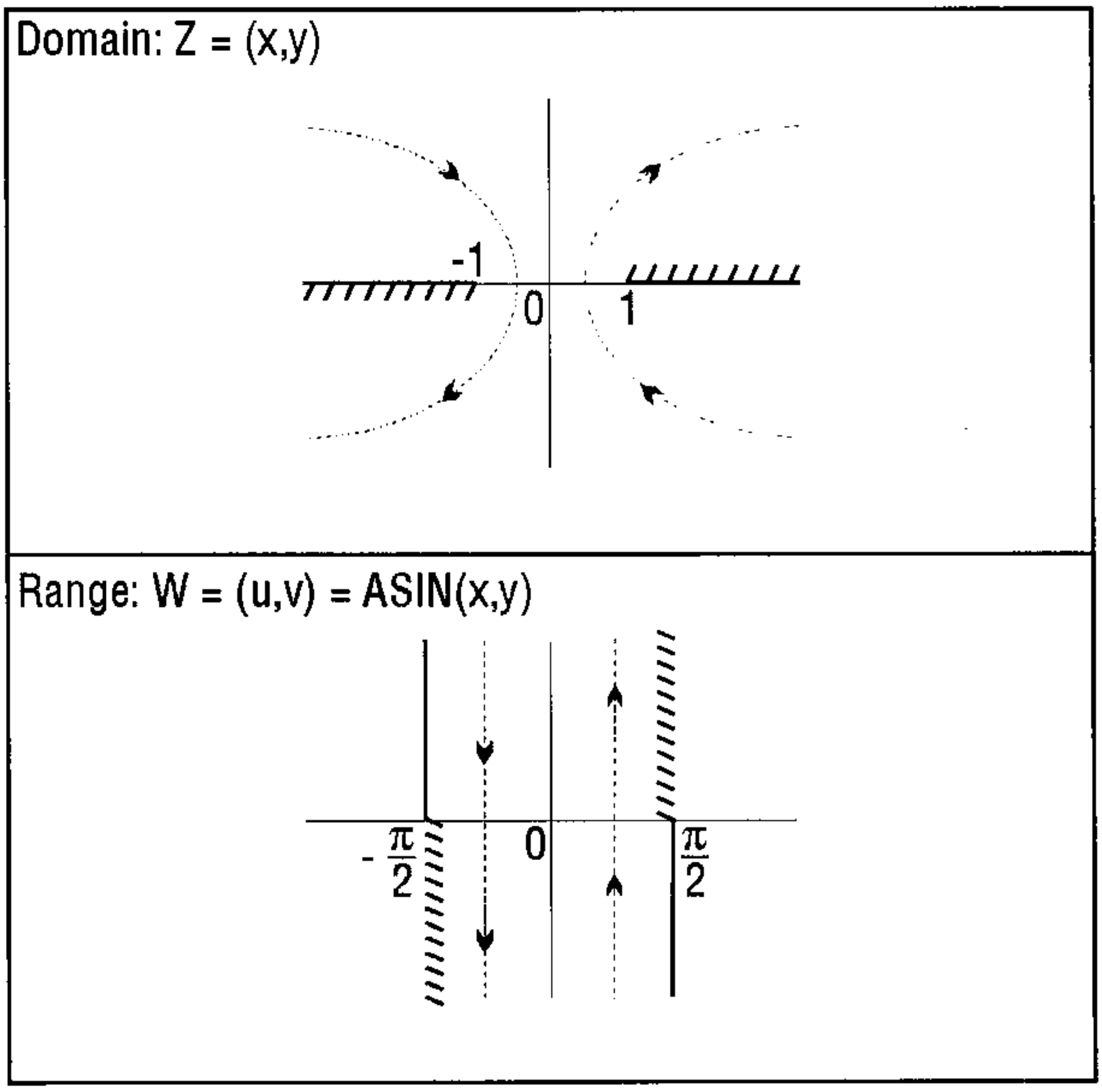
is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of ASIN. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

These graphs show the inverse relation 'ASIN(Z)+(-1)^n1+π*n1' for the case *n1*=0. For other values of *n1*, the vertical band in the lower graph is translated to the right (for *n1* positive) or to the left (for *n1* negative). Taken together, the bands cover the whole complex plane, which is the domain of SIN.

View these graphs with domain and range reversed to see how the domain of SIN is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (*x*, *y*). SIN sends this domain onto the whole complex plane in the range W = (*u*, *v*) = SIN(*x*, *y*) in the upper graph.

Related Commands: ACOS, ATAN, ISOL, SIN



Branch Cuts for ASIN(Z)

ASINH

Arc Hyperbolic Sine Analytic Function: Returns the inverse hyperbolic sine of the argument.

{}

Level 1	→	Level 1
z	→	asinh *z*
'*symb*'	→	'ASINH(*symb*)'

Keyboard Access: (MTH) HYP ASINH

Affected by Flags: Principal Solution (−1), Numerical Results (−3)

Remarks: The inverse of SINH is a *relation*, not a function, since SINH sends more than one argument to the same result. The inverse relation for SINH is expressed by ISOL as the *general solution*

```
'ASINH(Z)*(-1)^n1+π*i*n1'
```

The function ASINH is the inverse of a *part* of SINH, a part defined by restricting the domain of SINH such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of SINH are called the *principal values* of the inverse relation. ASINH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ASINH to the boundary of the restricted domain of SINH form the *branch cuts* of ASINH.

The principal branch used by the HP 48 for ASINH was chosen because it is analytic in the regions where the arguments of the *real-valued* function are defined. The branch cut for the complex-valued ASINH function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graph for ASINH can be found from the graph for ASIN (see ASIN) and the relationship asinh $z = -i$ asin iz.

Related Commands: ACOSH, ATANH, ISOL, SINH

ASN

Assign Command: Defines a single key on the user keyboard by assigning the given object to the key x_{key}, which is specified as $rc.p$.

{ }

Level 2	Level 1	→	Level 1
obj	x_{key}	→	
'SKEY'	x_{key}	→	

Keyboard Access: ◀ MODES KEYS ASN

Affected by Flags: User-Mode Lock (−61) and User Mode (−62) affect the status of the user keyboard

Remarks: The argument x_{key} is a real number $rc.p$ specifying the key by its row number r, column number c, and plane (shift) p. The legal values for p are as follows:

Plane, p	Shift
0 or 1	unshifted
2	◀ left-shifted
3	▶ right-shifted
4	α alpha-shifted
5	α ◀ alpha left-shifted
6	α ▶ alpha right-shifted

Once ASN has been executed, pressing a given key in User or 1-User mode executes the user-assigned object. The user key assignment remains in effect until the assignment is altered by ASN, STOKEYS, or DELKEYS. Keys without user assignments maintain their standard definitions.

If the argument *obj* is the name 'SKEY', then the specified key is restored to its *standard key* assignment on the user keyboard. This

is meaningful only when all standard key assignments had been suppressed (for the user keyboard) by the command '5' DELKEYS (see DELKEYS).

To make multiple key assignments simultaneously, use STOKEYS. To delete key assignments, use DELKEYS.

Be careful not to reassign or suppress the keys necessary to cancel User mode. If this happens, exit User mode by doing a system halt ("warm start"): press and hold ON and the C key simultaneously, releasing the C key first. This cancels User mode.

Example: Executing ASN with GETI in level 2 and 85.3 in level 1 assigns GETI to → " " on the user keyboard. (→ " " has a location of 85.3 because it is eight rows down, five columns across, and right-shifted.) When the calculator is in User mode, pressing → " " now executes GETI (instead of executing " ").

Related Commands: DELKEYS, RCLKEYS, STOKEYS

ASR

Arithmetic Shift Right Command: Shifts a binary integer one bit to the right, except for the most significant bit, which is maintained.

{ }

Level 1	→	Level 1
#n_1	→	#n_2

Keyboard Access: MTH BASE NXT BIT ASR

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The most significant bit is preserved while the remaining (*wordsize*−1) bits are shifted right one bit. The second-most significant bit is replaced with a zero. The least significant bit is shifted out and lost.

ASR

An arithmetic shift is useful for preserving the sign bit of a binary integer that will be shifted. Although the HP 48 makes no special provision for signed binary integers, you can still *interpret* a number as a signed quantity.

Related Commands: SL, SLB, SR, SRB

ATAN

Arc Tangent Analytic Function: Returns the value of the angle having the given tangent.

{ }

Level 1	→	Level 1
z	→	arc tan *z*
'*symb*'	→	'ATAN(*symb*)'

Keyboard Access: ◀ ATAN

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Angle Mode (−17, −18)

Remarks: For a real argument, the result ranges from −90 to +90 degrees ($-\pi/2$ to $+\pi/2$ radians; −100 to +100 grads).

The inverse of TAN is a *relation*, not a function, since TAN sends more than one argument to the same result. The inverse relation for TAN is expressed by ISOL as the *general solution*

```
'ATAN(Z)+π*n1'
```

The function ATAN is the inverse of a *part* of TAN, a part defined by restricting the domain of TAN such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of TAN are called the *principal values* of the inverse relation. ATAN in its entirety is called the *principal branch* of the inverse relation, and the points sent by ATAN to the boundary of the restricted domain of TAN form the *branch cuts* of ATAN.

The principal branch used by the HP 48 for ATAN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cuts for the complex-valued arc tangent function occur where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

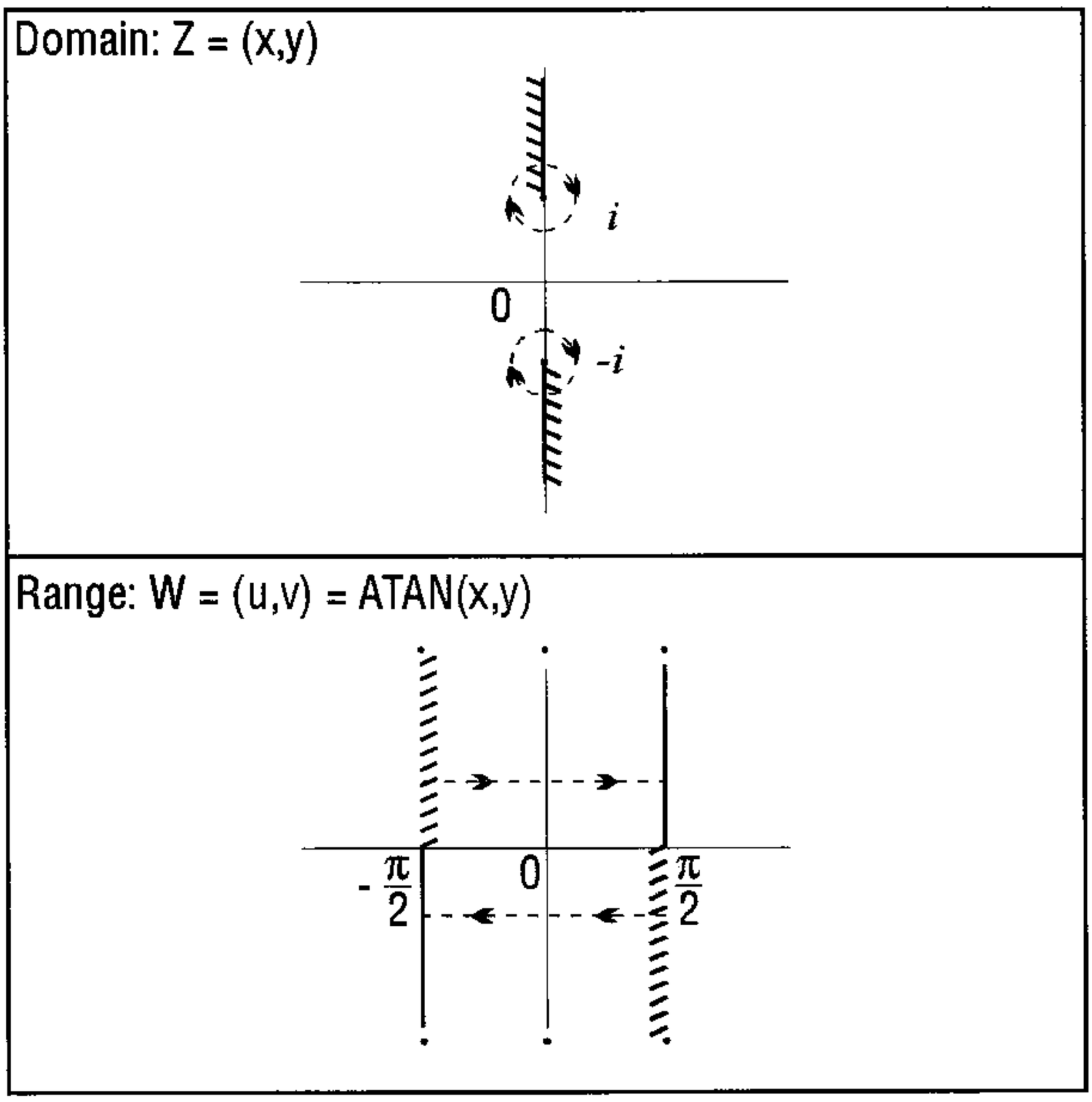
The graphs below show the domain and range of ATAN. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

These graphs show the inverse relation 'ATAN(Z)+π*n1' for the case *n1*=0. For other values of *n1*, the vertical band in the lower graph is translated to the right (for *n1* positive) or to the left (for *n1* negative). Taken together, the bands cover the whole complex plane, which is the domain of TAN.

View these graphs with domain and range reversed to see how the domain of TAN is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (*x*, *y*). TAN sends this domain onto the whole complex plane in the range W = (*u*, *v*) = TAN(*x*, *y*) in the upper graph.

Related Commands: ACOS, ASIN, ISOL, TAN

ATAN



Branch Cuts for ATAN(Z)

ATANH

Arc Hyperbolic Tangent Analytic Function: Returns the inverse hyperbolic tangent of the argument.

{ }

Level 1	→	Level 1
z	→	atanh *z*
'*symb*'	→	'ATANH(*symb*)'

Keyboard Access: (MTH) HYP ATANH

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Infinite Result Exception (−22)

Remarks: For real arguments $|x| > 1$, ATANH returns the complex result obtained for the argument $(x, 0)$. For a real argument $x=\pm 1$, an `Infinite Result` exception occurs. If flag -22 is set (no error), the sign of the result (MAXR) matches that of the argument.

The inverse of TANH is a *relation*, not a function, since TANH sends more than one argument to the same result. The inverse relation for TANH is expressed by ISOL as the *general solution*

```
'ATANH(Z)+π*i*n1'
```

The function ATANH is the inverse of a *part* of TANH, a part defined by restricting the domain of TANH such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of TANH are called the *principal values* of the inverse relation. ATANH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ATANH to the boundary of the restricted domain of TANH form the *branch cuts* of ATANH.

The principal branch used by the HP 48 for ATANH was chosen because it is analytic in the regions where the arguments of the *real-valued* function are defined. The branch cut for the complex-valued ATANH function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graph for ATANH can be found from the graph for ATAN (see ATAN) and the relationship atanh $z = -i$ atan iz.

Related Commands: ACOSH, ASINH, ISOL, TANH

ATICK

Axes Tick-Mark Command: Sets the axes tick-mark annotation in the reserved variable *PPAR*.

Level 1	→	Level 1
x	→	
#n	→	
{ *x,y* }	→	
{ *#n #m* }	→	

Keyboard Access: ◀ PLOT PPAR NXT ATICK

Affected by Flags: None

Remarks: Given x, ATICK sets the tick-mark annotation to x units on both the x- and the y-axis. For example, 2 would place tick-marks every 2 units on both axes.

Given $\#n$, ATICK sets the tick-mark annotation to $\#n$ pixels on both the x- and the y-axis. For example, #5 would place tick-marks every 5 pixels on both axes.

Given { *x y* }, ATICK sets the tick-mark unit annotation for each axis individually. For example, { 10 3 } would mark the x-axis at every multiple of 10 units, and the y-axis at every multiple of 3 units.

Given { *#n #m* } ATICK sets the tick-mark pixel annotation for each axis individually. For example, { 6 2 } would mark the x-axis every 6 pixels, and the y-axis every 2 pixels.

Related Commands: AXES, DRAX

ATTACH

Attach Library Command: Attaches the library with the specified number to the current directory. Each library has a unique number. If a port number is specified, it is ignored.

{ }

Level 1	→	Level 1
$n_{library}$	→	
$:n_{port}$ $:n_{library}$	→	

Keyboard Access: ◀ LIBRARY NXT ATTACH

Affected by Flags: None

Remarks: To use a library object, it must be in a port and it must be attached. A library object from an application card (ROM) is automatically in a port (1-33), but a library object copied into RAM (such as through the PC Link) must be stored into a port using STO.

Many libraries are attached automatically when an application card is installed. Others require you to ATTACH them, as do many libraries copied into RAM. (The owner's manual for the application card or library will tell you which of its library objects must be attached manually.) You can also ascertain whether a library is attached to the current directory by executing LIBS.

A library that has been copied into RAM and then stored (with STO) into a port can be attached *only after the calculator has been turned off and then on again* following the STO command. This action (off/on) creates a *system halt*, which makes the library object "attachable." Note that it also clears the stack, local variables, and the LAST stack, and it displays the MATH menu. (To save the stack first, execute DEPTH →LIST '*name*' STO .)

The number of libraries that can be attached to the HOME directory is limited only by the available memory. However, only one library at a time can be attached to any other directory. If you attempt to attach a second library to a non-*HOME* directory, the new library will overwrite the old one.

ATTACH

Related Commands: DETACH, LIBS

AUTO

Autoscale Command: Calculates a y-axis display range, or an x- and y-axis display range.

Keyboard Access: ◀ PLOT NXT AUTO

Affected by Flags: None

Remarks: The action of AUTO depends on the plot type as follows:

Plot Type	Scaling Action
FUNCTION	Samples the equation in EQ at 40 values of the independent variable, equally spaced through the x-axis plotting range, discards points that return $\pm\infty$, then sets the y-axis display range to include the maximum, minimum, and origin.
CONIC	Sets the y-axis scale equal to the x-axis scale.
POLAR	Samples the equation in EQ at 40 values of the independent variable, equally spaced through plotting range, discards points that return $\pm\infty$, then sets both the x- and y-axis display ranges in the same manner as for plot type FUNCTION.
PARAMETRIC	Same as POLAR.
TRUTH	No action.
BAR	Sets the x-axis display range from 0 to the number of elements in ΣDAT, plus 1. Sets the y-range to the minimum and maximum of the elements. The x-axis is always included.

Plot Type	Scaling Action
HISTOGRAM	Sets the x-axis display range to the minimum and maximum of the elements in ΣDAT. Sets the y-axis display range from 0 to the number of rows in ΣDAT.
SCATTER	Sets the x-axis display range to the minimum and maximum of the independent variable column (XCOL) in ΣDAT. Sets the y-axis display range to the minimum and maximum of the dependent variable column (YCOL).

AUTO does not affect 3D plots.

AUTO actually calculates a y-axis display range and then expands that range so that the menu labels do not obscure the resultant plot.

AUTO does not draw a plot—execute DRAW to do so.

Example: The program « FUNCTION AUTO DRAW DRAX » sets the plot type to FUNCTION, autoscales the y-axis, plots the equation in EQ, and adds axes to the plot.

Related Commands: DRAW, *H, SCALE, SCLΣ, *W, XRNG, YRNG

AXES

Axes Command: Specifies the intersection coordinates of the x- and y-axes, tick-mark annotation, and the labels for the x- and y-axes. This information is stored in the reserved variable $PPAR$.

Level 1	→	Level 1
(*x*, *y*)	→	
{ (*x*, *y*) *atick* "*x-axis label*" "*y-axis label*" }	→	

AXES

Keyboard Access: ◀ PLOT PPAR NXT AXES

Affected by Flags: None

Remarks: The argument for AXES (a complex number or list) is stored as the fifth parameter in the reserved variable *PPAR*. How the argument is used depends on the type of object it is:

- If the argument is a complex number, it replaces the current entry in *PPAR*.
- If the argument is a list containing any or all of the above variables, only variables that are specified are affected.

atick has the same format as the argument for the ATICK command. This is the variable that is affected by the ATICK command.

The default value for AXES is (0,0).

Axes labels are not displayed in *PICT* until subsequent execution of LABEL.

Example: The command sequence

```
{ (0,0) 2 "t" "y" } AXES LABEL
```

specifies an axes intersection at (0,0), tick-mark annotation every 2 units, and puts the labels t and y in *PICT*. The labels are positioned to identify the horizontal and vertical axes respectively.

Related Commands: ATICK, DRAW, DRAX, LABEL

BAR

Bar Plot Type Command: Sets the plot type to BAR.

Keyboard Access: ◀ PLOT NXT STAT PTYPE BAR

Affected by Flags: None

Remarks: When the plot type is BAR, the DRAW command plots a bar chart using data from one column of the current statistics matrix (reserved variable ΣDAT). The column to be plotted is specified by the XCOL command, and is stored in the first parameter of the reserved variable ΣPAR. The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

$\{\ \langle x_{min},\ y_{min}\rangle\ \langle x_{max},\ y_{max}\rangle\ indep\ res\ axes\ ptype\ depend\ \}$

For plot type BAR, the elements of *PPAR* are used as follows:

- $\langle x_{\min}, y_{\min}\rangle$ is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (-6.5,-3.1).
- $\langle x_{\max}, y_{\max}\rangle$ is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is either a name specifying a label for the horizontal axis, or a list containing such a name and two numbers, with the smaller of the numbers specifying the horizontal location of the first bar. The default value of *indep* is *X*.
- *res* is a real number specifying the bar width in user-unit coordinates, or a binary integer specifying the bar width in pixels. The default value is 0, which specifies a bar width of 1 in user-unit coordinates.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is {0,0}.
- *ptype* is a command name specifying the plot type. Executing the command BAR places the command name BAR in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

A bar is drawn for each element of the column in ΣDAT. Its width is specified by *res* and its height is the value of the element. The location of the first bar can be specified by *indep*; otherwise, the value in $\langle x_{\min}, y_{\min}\rangle$ is used.

Related Commands: CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

BARPLOT

Draw Bar Plot Command: Plots a bar chart of the specified column of the current statistics matrix (reserved variable ΣDAT).

Keyboard Access: ↰ STAT PLOT BARPLOT

Affected by Flags: None

Remarks: The data column to be plotted is specified by XCOL and is stored as the first parameter in reserved variable ΣPAR. The default column is 1. Data can be positive or negative, resulting in bars above or below the axis. The y-axis is autoscaled, and the plot type is set to BAR.

When BARPLOT is executed from a program, the resulting plot does not persist unless PICTURE, PVIEW (with an empty list argument), or FREEZE is subsequently executed.

Related Commands: FREEZE, HISTPLOT, PICTURE, PVIEW, SCATRPLOT, XCOL

BAUD

Baud Rate Command: Specifies bit-transfer rate.

{ }

Level 1	→	Level 1
n_{baudrate}	→	

Keyboard Access: ↰ I/O IOPAR BAUD

Affected by Flags: None

Remarks: Legal baud rates are 1200, 2400, 4800, and 9600 (default).

For more information, refer also to the reserved variable *IOPAR* (*I/O parameters*) in appendix D, "Reserved Variables."

Related Commands: CKSM, PARITY, TRANSIO

BEEP

Beep Command: Sounds a tone at n hertz for x seconds.

{ }

Level 2	Level 1	→	Level 1
$n_{frequency}$	$x_{duration}$	→	

Keyboard Access: PRG NXT UT NXT BEEP

Affected by Flags: Error Beep (−56)

Remarks: The frequency of the tone is subject to the resolution of the built-in tone generator. The maximum frequency is approximately 4400 Hz; the maximum duration is 1048.575 seconds. Arguments greater than these maximum values default to the maxima.

Related Commands: HALT, INPUT, PROMPT, WAIT

BESTFIT

Best-Fitting Model Command: Executes LR with each of the four curve fitting models, and selects the model yielding the largest correlation coefficient.

Keyboard Access: STAT ΣPAR MODL BESTF

Affected by Flags: None

Remarks: The selected model is stored as the fifth parameter in the reserved variable ΣPAR, and the associated regression coefficients, intercept and slope, are stored as the third and fourth parameters, respectively. For a description of ΣPAR, see appendix D, "Reserved Variables."

Related Commands: EXPFIT, LINFIT, LOGFIT, LR, PWRFIT

BIN

Binary Mode Command: Selects binary base for binary integer operations. (The default base is decimal.)

Keyboard Access: (MTH) BASE BIN

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: Binary integers require the prefix #. Binary integers entered and returned in binary base automatically show the suffix ⊟. If the current base is not binary, binary numbers can still be entered by using the suffix ⊟ (the numbers are displayed in the current base, however).

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Related Commands: DEC, HEX, OCT, STWS, RCWS

BINS

Sort Into Frequency Bins Command: Sorts the elements of the independent column (XCOL) of the current statistics matrix (the reserved variable ΣDAT) into $(n_{\text{bins}} + 2)$ bins, where the left edge of bin 1 starts at value x_{min} and each bin has width x_{width}.

{ }

Level 3	Level 2	Level 1	→	Level 2	Level 1
x_{min}	x_{width}	n_{bins}	→	[[$n_{\text{bin}1}$ ⋯ $n_{\text{bin}n}$]]	[n_{binL} n_{binR}]

Keyboard Access: ↩ (STAT) 1-VAR BINS

Affected by Flags: None

Remarks: BINS returns a matrix containing the frequency of occurrences in each bin, and a 2-element array containing the frequency of occurrences falling below or above the defined range of x-values. The array can be stored into the reserved variable ΣDAT

and used to plot a bar histogram of the bin data (for example, by executing BARPLOT).

For each element x in ΣDAT, the nth bin count $n_{\text{freq bin n}}$ is incremented, where:

$$n_{freq\ bin\ n} = IP\left[\frac{x - x_{min}}{x_{width}}\right]$$

for $x_{\text{min}} \leq x \leq x_{\text{max}}$, where $x_{\text{max}} = x_{\text{min}} + (n_{\text{bins}})(x_{\text{width}})$.

Example: If the independent column of ΣDAT contains the following data:

7 2 8 1 4 6 9 0 1 1 8 5 13 2 6 9 5 8 5

1 2 5 BINS returns [[5] [3] [5] [2] [2]] and [1 1].

The data has been sorted into 5 bins of width 2, starting at x-value 1 and ending at x-value 11. The first element of the matrix shows that 5 x-values (2 1 1 1 2) fell in bin 1, where bin 1 ranges from x-value 1 through 2.99999999999. The vector shows that one x-value was less than x_{min} (0), and one was greater than x_{max} (13).

Related Commands: BARPLOT, XCOL

BLANK

Blank Graphics Object Command: Creates a blank graphics object of the specified width and height.

{ }

Level 2	Level 1	→	Level 1
$\#n_{\text{width}}$	$\#m_{\text{height}}$	→	$grob_{\text{blank}}$

Keyboard Access: PRG GROB BLANK

Affected by Flags: None

Related Commands: →GROB, LCD→

BOX

Box Command: Draws in *PICT* a box whose opposite corners are defined by the specified pixel or user-unit coordinates.

{ }

Level 2	Level 1	→	Level 1
{ #n_1 #m_1 }	{ #n_2 #m_2 }	→	
(x_1 , y_1)	(x_2 , y_2)	→	

Keyboard Access: PRG PICT BOX

Affected by Flags: None

Related Commands: ARC, LINE, TLINE

BUFLEN

Buffer Length Command: Returns the number of characters in the HP 48's serial input buffer and a single digit indicating whether an error occurred during data reception.

Level 1	→	Level 2	Level 1
	→	n_{chars}	0/1

Keyboard Access: ← I/O NXT SERIA BUFLE

Affected by Flags: None

Remarks: The digit returned to level 1 is 1 if no framing, UART overrun, or input-buffer overflow errors occurred during reception, or 0 if one of these errors did occur. (The input buffer holds up to 255 bytes.) When a framing or overrun error occurs, data reception ceases until the error is cleared (which BUFLEN does); therefore, *n* represents the data received *before* the error.

Use ERRM to see which error has occurred when BUFLEN returns 0 to level 1.

Related Commands: CLOSEIO, OPENIO, SBRK, SRECV, STIME, XMIT

BYTES

Byte Size Command: Returns the number of bytes and the checksum for the given object.

{ }

Level 1	→	Level 2	Level 1
obj	→	#$n_{checksum}$	x_{size}

Keyboard Access: ← MEMORY BYTES

Affected by Flags: None

Remarks: If the argument is a built-in object, then the size is 2.5 bytes and the checksum is # 0.

If the argument is a global name, then the size represents the name *and* its contents, while the checksum represents the contents only. The size of the name alone is $(3.5 + 2 \times n)$, where n is the number of characters in the name.

Example: Objects that decompile identically can have different byte sizes and checksums. For instance,

```
« 1 »
```

and

```
1 'A' STO A « » +
```

both produce lists containing the number 1. However, the first list contains the built-in object 1 (for a size of 7.5 bytes), while the second list contains a RAM copy of 1 (for a size of 15.5 bytes).

Related Commands: MEM

B→R

Binary to Real Command: Converts a binary integer to its floating-point equivalent.

{}

Level 1	→	Level 1
#*n*	→	*n*

Keyboard Access: (MTH) BASE B→R

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: If # $n \geq$ # 1000000000000 (base 10), only the 12 most significant decimal digits are preserved in the resulting mantissa.

Related Commands: R→B

CASE

CASE Conditional Structure Command: Starts CASE ... END conditional structure.

	Level 1	→	Level 1
CASE		→	
THEN	*T/F*	→	
END		→	
END		→	

Keyboard Access: (PRG) BRCH CASE CASE

Affected by Flags: None

Remarks: The CASE ... END structure executes a series of *cases* (tests). The first test that returns a true result causes execution of the corresponding true-clause, ending the CASE ... END structure. A default clause can also be included: this clause executes if all tests evaluate to false.

The CASE ... END structure has this syntax:

```
CASE
        test-clause1 THEN true-clause1 END
        test-clause2 THEN true-clause2 END
        :
        test-clausen THEN true-clausen END
        default-clause (optional)
END
```

When CASE executes, *test-clause*$_1$ is evaluated. If the test is true, *true-clause*$_1$ executes, then execution skips to END. If *test-clause*$_1$ is false, *test-clause*$_2$ executes. Execution within the CASE structure continues until a true clause is executed, or until all the test clauses evaluate to false. If the default clause is included, it executes if all test clauses evaluate to false.

Example: The following program takes a numeric argument from the stack:

- if the argument is negative, it is added to itself
- if the argument is positive, it is negated
- if the argument is zero, the program aborts

```
« → X
  « CASE
    'X>0'
    THEN X NEG END
    'X<0'
    THEN X DUP + END
    'X==0'
    THEN 0 DOERR END
END
```

CASE

»

»

Related Commands: END, IF, IFERR, THEN

CEIL

Ceiling Function: Returns the smallest integer greater than or equal to the argument.

Level 1	→	Level 1
x	→	*n*
x_unit	→	*n_unit*
'*symb*'	→	'CEIL(*symb*)'

Keyboard Access: MTH REAL NXT NXT CEIL

Affected by Flags: Numerical Results (−3)

Examples: 3.2 CEIL returns 4; −3.2 CEIL returns −3.

Related Commands: FLOOR, IP, RND, TRNC

CENTR

Center Command: Adjusts the first two parameters in the reserved variable *PPAR*, (x_{min}, y_{min}) and (x_{max}, y_{max}), so that the point represented by the argument (*x*, *y*) is the plot center.

{ }

Level 1	→	Level 1
(*x*, *y*)	→	
x	→	

Keyboard Access: ◀ PLOT FFHF NXT CENT

Affected by Flags: None

Remarks: The center pixel is in row 32, column 65 when *PICT* is its default size (131 × 64).

If the argument is a real number x, CENTR makes the point $(x, 0)$ the plot center.

Related Commands: SCALE

CF

Clear Flag Command: Clears the specified user or system flag.

{ }

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	

Keyboard Access:

◀ MODES FLAG CF

PRG TEST NXT NXT CF

Affected by Flags: None

Remarks: User flags are numbered 1 through 64. System flags are numbered −1 through −64. See appendix C, "System Flags," for a listing of HP 48 system flags and their flag numbers.

Related Commands: FC?, FC?C, FS?, FS?C, SF

CHOOSE

Create User-Defined Choose Box Command: Creates a user-defined choose box.

Level 3	Level 2	Level 1	→	Level 2	Level 1
"*prompt*"	{ c_1 … c_n }	n_{pos}	→	*obj* or *result*	"*1*"
"*prompt*"	{ c_1 … c_n }	n_{pos}	→		"*0*"

Keyboard Access: PRG NXT IN CHOOS

Affected by Flags: None

Remarks: CHOOSE creates a standard single-choice choose box based on the following specifications.

Variable	Function
"*prompt*"	A message that appears at the top of choose box. If "*prompt*" is an empty string (""), no message is displayed.
$\{c_1 \ \ldots \ c_n\}$	Definitions that appear within the choose box. A choice definition (c_x) can have two formats. ■ *obj*, any object. ■ { $obj_{display}$ obj_{result} }, the object to be displayed followed by the result returned to the stack if that object is selected.
n_{pos}	The position number of an item definition. This item is highlighted when the choose box appears. If n_{pos}=0, no item is highlighted, and the choose box can be used to view items only.

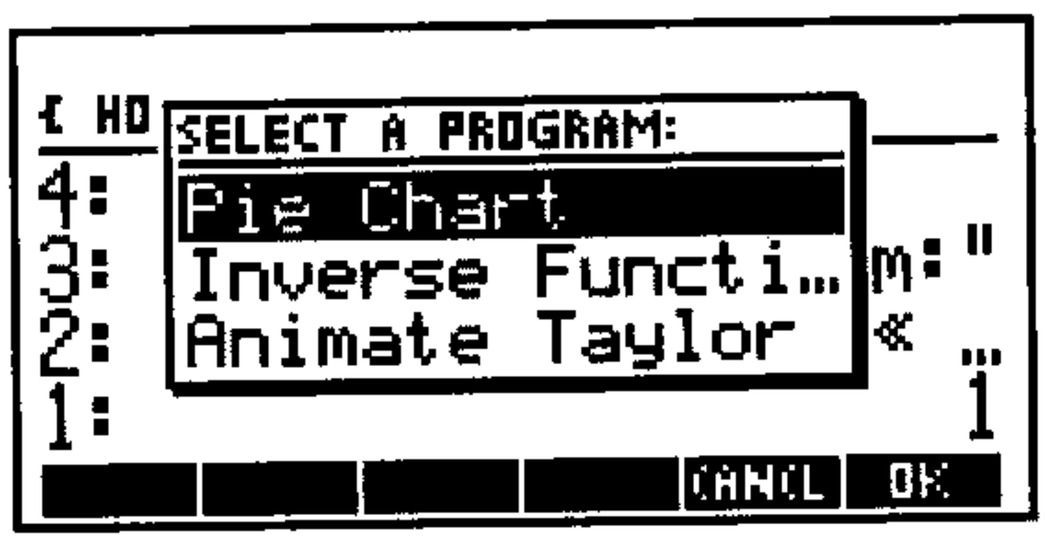
If you choose an item from the choose box and press OK, CHOOSE returns the *result* (or the object itself if no result is

specified) to level 2 and 1 to level 1. If you press CHNCL, CHOOSE returns 0. Also, if n_{pos}=0, CHOOSE returns 0.

Example: CHOOSE with the following three lines on the display.

```
"Select a Program:"
{ {"Pie Chart" «PIE»} {"Inverse Function" «ROOTR»}
{"Animate Taylor" «TSA»} }
1
```

would produce the following display.



Related Commands: INFORM, NOVAL

%CH

Percent Change Function: Returns the percent change from x (level 2) to y (level 1) as a percentage of x.

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	*100(y−x)/x*
x	'*symb*'	→	'%CH(*x*,*symb*)'
'*symb*'	*x*	→	'%CH(*symb*,*x*)'
'$symb_1$'	'$symb_2$'	→	'%CH($symb_1$,$symb_2$)'
x_unit	*y_unit*	→	*100(y_unit−x_unit)/x_unit*
x_unit	'*symb*'	→	'%CH(*x_unit*,*symb*)'
'*symb*'	*x_unit*	→	'%CH(*symb*,*x_unit*)'

%CH

Keyboard Access: (MTH) REAL %CH

Affected by Flags: Numerical Results (−3)

Remarks: If both arguments are unit objects, the units must be consistent with each other. The dimensions of a unit object are dropped from the result, *but units are part of the calculation.*

For more information on using temperature units with arithmetic functions, refer to the keyword entry for +.

Examples: 1_m 500_cm %CH returns 400, because 500 cm represents an increase of 400% over 1 m.

100_K 150_K %CH returns 50.

Related Commands: %, %T

CHR

Character Command: Returns a string representing the HP 48 character corresponding to the character code *n*.

{ }

Level 1	→	Level 1
n	→	*"string"*

Keyboard Access:

(←) (CHARS) CHR

(PRG) TYPE (NXT) CHR

Affected by Flags: None

Remarks: The character codes are an extension of ISO 8859/1. Codes 128 through 159 are unique to the HP 48. See the entry for NUM for a complete list of characters and character codes.

The default character ▪ is supplied for all character codes that are *not* part of the normal HP 48 display character set.

Character code 0 is used for the special purpose of marking the end of the command line. Attempting to edit a string containing this character causes the error Can't Edit CHR(0).

You can use the CHARS application to find the character code for any character used by the HP 48. See "Keying in Special Characters" in chapter 2 of the *HP 48 User's Guide*.

Related Commands: NUM, POS, REPL, SIZE, SUB

CKSM

Checksum Command: Specifies the error-detection scheme.

{ }

Level 1	→	Level 1
n_{checksum}	→	

Keyboard Access: ◀ I/O IOFHR CKSM

Affected by Flags: None

Remarks: Legal values for n_{checksum} are as follows:

- 1: 1-digit arithmetic checksum.
- 2: 2-digit arithmetic checksum.
- 3: 3-digit cyclic redundancy check (default).

The CKSM specified is the error-detection scheme that will be requested by KGET, PKT, or SEND. If the sender and receiver disagree about the request, however, then a 1-digit arithmetic checksum will be used.

IR transmission should use checksum type 3.

Related Commands: BAUD, PARITY, TRANSIO

CLEAR

Clear Command: Removes all objects from the stack.

Level n ... Level 1	→	Level n ... Level 1
obj_n ... obj_1	→	

Keyboard Access: ◀ CLEAR

Affected by Flags: None

Remarks: To recover a cleared stack, press ▶ UNDO before executing any other operation. There is no programmable command to recover the stack.

Related Commands: CLVAR, PURGE

CLKADJ

Adjust System Clock Command: Adjusts the system time by x clock ticks, where 8192 clock ticks equal 1 second.

{ }

Level 1	→	Level 1
x	→	

Keyboard Access: ◀ TIME NXT NXT CLKH

Affected by Flags: None

Remarks: If x is positive, x clock ticks are added to the system time. If x is negative, x clock ticks are subtracted from the system time.

Example: -20480 CLKADJ decrements the system time by 2.5 seconds.

Related Commands: →TIME

CLLCD

Clear LCD Command: Clears (blanks) the stack display.

Keyboard Access: PRG NXT CUT CLLCD

Affected by Flags: None

Remarks: The menu labels continue to be displayed after execution of CLLCD.

When executed from a program, the blank display persists only until the keyboard is ready for input. To cause the blank display to persist until a key is pressed, execute FREEZE after executing CLLCD. (When executed from the keyboard, CLLCD *automatically* freezes the display.)

Example: Evaluating « CLLCD 7 FREEZE » blanks the display (except the menu labels), then freezes the entire display.

Related Commands: DISP, FREEZE

CLOSEIO

Close I/O Port Command: Closes the serial port and the IR port, and clears the input buffer and any error messages for KERRM.

Keyboard Access: ← I/O NXT CLOSE

Affected by Flags: None

Remarks: When the HP 48 turns off, it automatically closes the serial and IR ports, but does not clear KERRM. Therefore, CLOSEIO is not needed to close the ports, but can conserve power without turning off the calculator.

Executing HP 48 Kermit protocol commands automatically clears the input buffer; however, executing non-Kermit commands (such as SRECV and XMIT) does not.

CLOSEIO

CLOSEIO also clears error messages from KERRM. This can be useful when debugging.

Related Commands: BUFLEN, OPENIO

CLΣ

Clear Sigma Command: Purges the current statistics matrix (reserved variable ΣDAT).

Keyboard Access: ◀ STAT DATA CLΣ

Affected by Flags: None

Related Commands: RCLΣ, STOΣ, Σ+, Σ−

CLTEACH

Clear Teaching Examples Command: Removes the EXAMPLES subdirectory and its contents from the HOME directory.

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Related Commands: TEACH

CLUSR

Clear Variables Command: Provided for compatibility with the HP 28. CLUSR is the same as CLVAR. See CLVAR.

CLVAR

Clear Variables Command: Purges all variables and empty subdirectories in the current directory.

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Related Commands: CLUSR, PGDIR, PURGE

CNRM

Column Norm Command: Returns the column norm (one-norm) of the array argument.

{ }

Level 1	→	Level 1
[*array*]	→	$x_{columnnorm}$

Keyboard Access: (MTH) MATR NORM CNRM

Affected by Flags: None

Remarks: The column norm of a matrix is the maximum (over all columns) of the sum of the absolute values of all elements in each column. For a vector, the column norm is the sum of the absolute values of the vector elements. For complex arrays, the absolute value of a given element (x, y) is $\sqrt{x^2 + y^2}$.

Related Commands: CROSS, DET, DOT, RNRM

→COL

Matrix to Columns Command: Transforms a matrix into a series of column vectors and returns the vectors and a column count, or transforms a vector into its elements and returns the elements and an element count.

{ }

Level 1	→	Level n+1 ...	Level 2	Level 1
[[*matrix*]]	→	[*vector*]$_{col1}$	[*vector*]$_{coln}$	$n_{colcount}$
[*vector*]	→	$element_1$	$element_n$	$n_{elementcount}$

Keyboard Access: (MTH) MATR COL →COL

Affected by Flags: None

Remarks: →COL introduces no rounding error.

Related Commands: COL→, →ROW, ROW→

COL+

Insert Column Command: Inserts an array (vector or matrix) into a matrix (or one or more elements into a vector) at the position indicated by n_{index}, and returns the modified array.

{ }

Level 3	Level 2	Level 1	→	Level 1
[[*matrix*]]$_1$	[*matrix*]$_2$	n_{index}	→	[[*matrix*]]$_3$
[[*matrix*]]$_1$	[*vector*]$_{column}$	n_{index}	→	[[*matrix*]]$_2$
[*vector*]$_1$	$n_{element}$	n_{index}	→	[*vector*]$_2$

Keyboard Access: (MTH) MATR COL COL+

Affected by Flags: None

Remarks: The inserted array must have the same number of rows as the target array.

n_{index} is rounded to the nearest integer. The original array is redimensioned to include the new columns or elements, and the elements at and to the right of the insertion point are shifted to the right.

Related Commands: COL–, CSWP, ROW+, ROW–

COL–

Delete Column Command: Deletes column n of a matrix (or element n of a vector), and returns the modified matrix (or vector) and the deleted column (or element).

{ }

Level 2	Level 1	→	Level 2	Level 1
[[*matrix*]]$_1$	n_{column}	→	[[*matrix*]]$_2$	[*vector*]$_{column}$
[*vector*]$_1$	$n_{element}$	→	[*vector*]$_2$	*element*$_n$

Keyboard Access: MTH MATR COL COL–

Affected by Flags: None

Remarks: n is rounded to the nearest integer.

Related Commands: COL+, CSWP, ROW+, ROW–

COL→

Columns to Matrix Command: Transforms a series of column vectors and a column count into a matrix containing those columns, or transforms a sequence of numbers and an element count into a vector with those numbers as elements.

Level n+1 ...	Level 2	Level 1	→	Level 1
[*vector*]$_{\text{columnn1}}$	[*vector*]$_{\text{columnn}}$	$n_{\text{columncount}}$	→	[[*matrix*]]
$element_1$	$element_n$	$n_{\text{elementcount}}$	→	[*vector*]

Keyboard Access: MTH MATR COL COL→

Affected by Flags: None

Remarks: All vectors must have the same length. The column or element count is rounded to the nearest integer.

Related Commands: →COL, →ROW, ROW→

COLCT

Collect Like Terms Command: Simplifies an algebraic expression or equation by "collecting" like terms.

{ }

Level 1	→	Level 1
'$symb_1$'	→	'$symb_2$'
x	→	x
(x, y)	→	(x, y)

Keyboard Access: ← SYMBOLIC COLCT

Affected by Flags: None

Remarks: COLCT operates separately on the two sides of an equation, so that like terms on opposite sides of the equation are not combined.

Examples: '6+EXP(10)' COLCT returns 8.7182818284 6.

'5+X+9' COLCT returns '14+X'.

'X*1_m+X*9_cm' COLCT returns '(109_cm)*X'.

'X^2*Y*X^T*Y' COLCT returns 'X^(T+2)*Y^2'.

'X+3*X+Y+Y' COLCT returns '4*X+2*Y'.

Related Commands: EXPAN, ISOL, QUAD, SHOW

COLΣ

Sigma Columns Command: Specifies the independent-variable and dependent-variable columns of the current statistics matrix (the reserved variable ΣDAT).

{ }

Level 2	Level 1	→	Level 1
x_{xcol}	x_{ycol}	→	

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: COLΣ combines the functionality of XCOL and YCOL. It is included in the HP 48 for compatibility with the HP 28S.

The independent-variable column number x_{xcol} is stored as the first parameter in the reserved variable ΣPAR (the default is 1). The dependent-variable column number x_{ycol} is stored as the second parameter in ΣPAR (the default is 2).

COLΣ accepts and stores noninteger real numbers, but subsequent commands that use these two parameters in ΣPAR will cause errors.

COLΣ

Example: 2 5 COLΣ sets column 2 in ΣDAT as the independent-variable column, sets column 5 as the dependent-variable column, and stores 2 and 5 as the first and second elements in ΣPAR.

Related Commands: BARPLOT, BESTFIT, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, XCOL, YCOL

COMB

Combinations Function: Returns the number of possible combinations of n items taken m at a time.

{}

Level 2	Level 1	→	Level 1
n	m	→	$C_{n\,;m}$
'$symb_n$'	m	→	'COMB($symb_n$, m)'
n	'$symb_m$'	→	'COMB(n, $symb_m$)'
'$symb_n$'	'$symb_m$'	→	'COMB($symb_n$, $symb_m$)'

Keyboard Access: MTH NXT PROB COMB

Affected by Flags: Numerical Results (−3)

Remarks: The following formula is used to calculate $C_{n,m}$.

$$C_{n,m} = \frac{n!}{m!(n-m)!}$$

The arguments n and m must each be less than 10^{12}.

Related Commands: PERM, !

CON

Constant Array Command: Returns a constant array, defined as an array whose elements all have the same value.

Level 2	Level 1	→	Level 1
{ $n_{columns}$ }	$z_{constant}$	→	[$vector_{constant}$]
{ n_{rows} $m_{columns}$ }	$z_{constant}$	→	[[$matrix_{constant}$]]
[*R-array*]	$x_{constant}$	→	[$R\text{-}array_{constant}$]
[*C-array*]	$z_{constant}$	→	[$C\text{-}array_{constant}$]
'*name*'	$z_{constant}$	→	

Keyboard Access: (MTH) MATR MAKE CON

Affected by Flags: None

Remarks: The constant value is a real or complex number taken from level 1. The resulting array is either a new array, or an existing array with its elements replaced by the constant, depending on the object in level 2.

- **Creating a new array:** If level 2 contains a list of one or two integers, CON returns a new array. If the list contains a single integer $n_{columns}$, CON returns a constant vector with n elements. If the list contains two integers n_{rows} and $m_{columns}$, CON returns a constant matrix with n rows and m columns.
- **Replacing the elements of an existing array:** If level 2 contains an array, CON returns an array of the same dimensions, with each element equal to the constant. If the constant is a complex number, the original array must also be complex.

  If level 2 contains a name, the name must identify a variable that contains an array. In this case, the elements of the array are replaced by the constant. If the constant is a complex number, the original array must also be complex.

Examples: { 2 2 } 6 CON returns the matrix [[6 6][6 6]].

[(2,4) (7,9)] 3 CON returns the complex vector
[(3,0) (3,0)].

CON

Related Commands: IDN

COND

Condition Number Command: Returns the 1-norm (column norm) condition number of a square matrix.

{ }

Level 1	→	Level 1
[[*matrix*]]$_{n \times n}$	→	$x_{conditionnumber}$

Keyboard Access: (MTH) MATR NORM COND

Affected by Flags: None

Remarks: The condition number of a matrix is the product of the norm of the matrix and the norm of the inverse of the matrix. COND uses the 1-norm and computes the condition number of the matrix without computing the inverse of the matrix.

The condition number expresses the sensitivity of the problem of solving a system of linear equations having coefficients represented by the elements of the matrix (this includes inverting the matrix). That is, it indicates how much an error in the inputs may be magnified in the outputs of calculations using the matrix.

In many linear algebra computations, the base 10 logarithm of the condition number of the matrix is an estimate of the number of digits of precision that might be lost in computations using that matrix. A reasonable rule of thumb is that the number of digits of accuracy in the result is approximately MIN(12,15−$\log_{10}$(COND)).

Example: The following program computes the above rule of thumb for the number of accurate digits:

```
«
  DUP SIZE 1 GET LOG SWAP COND LOG + 11 SWAP -
»
```

Related Commands: SNRM, SRAD, TRACE

CONIC

Conic Plot Type Command: Sets the plot type to CONIC.

Keyboard Access: ◀ PLOT FTYPE CONIC

Affected by Flags: None

Remarks: When the plot type is CONIC, the DRAW command plots the current equation as a second-order polynomial of two real variables. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

For plot type CONIC, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (-6.5,-3.1).

- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).

- *indep* is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value is *X*.

- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.

- *axes* is a complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes, or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).

CONIC

- *ptype* is a command name specifying the plot type. Executing the command CONIC places the command name CONIC in *PPAR*.
- *depend* is a name specifying the dependent variable. The default value is Y.

The current equation is used to define a pair of functions of the independent variable. These functions are derived from the second-order Taylor's approximation to the current equation. The minimum and maximum values of the independent variable (the plotting range) can be specified in *indep*; otherwise, the values in $\{x_{min}, y_{min}\}$ and $\{x_{max}, y_{max}\}$ (the display range) are used. Lines are drawn between plotted points unless flag −31 is set.

Related Commands: BAR, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

CONJ

Conjugate Analytic Function: Conjugates a complex number or a complex array.

{ }

Level 1	→	Level 1
x	→	x
(x, y)	→	$(x, -y)$
[*R-array*]	→	[*R-array*]
[*C-array*]$_1$	→	[*C-array*]$_2$
'*symb*'	→	'CONJ(*symb*)'

Keyboard Access: MTH NXT CMPL NXT CONJ

Affected by Flags: Numerical Results (−3)

Remarks: Conjugation is the negation (sign reversal) of the imaginary part of a complex number. For real numbers and real arrays, the conjugate is identical to the original argument.

Example: `[ (3,4) (7,2) ] CONJ` returns `[ (3,-4) (7,-2) ]`.

A square matrix A containing complex elements is said to be *Hermitian* if $A^H = A$, where A^H is the same as a normal transpose except that the complex conjugate of each element is used. The following program returns 1 if the input matrix is Hermitian, and a 0 if it is not.

```
« DUP TRN CONJ SAME »
```

Related Commands: ABS, IM, RE, SCONJ, SIGN

CONLIB

Open Constants Library Command: Opens the Constants Library catalog.

Keyboard Access: ← EQ LIB `CONLIB CONLI`

Affected by Flags: None

Related Commands: CONST

CONST

Constant Value Command: Returns the value of a constant.

{}

Level 1	→	Level 1
'name'	→	*x*

Keyboard Access: ← EQ LIB `CONLIB CONS`

Affected by Flags: Units Type (60), Units Usage (61)

CONST

Remarks: CONST returns the value of the specified constant. It chooses the unit type depending on flag 60 (SI if clear, English if set), and uses the units depending on flag 61 (uses units if clear, no units if set).

See "Using the Constants Library" in chapter 25 of the *HP 48 User's Guide* for a list of the constants available in the HP 48's Constants Library.

Related Commands: CONLIB

CONT

Continue Program Execution Command: Resumes execution of a halted program.

Keyboard Access: ← CONT

Affected by Flags: None

Remarks: Since CONT is a command, it can be assigned to a key or to a custom menu.

Example: The program

```
« "Enter A, press ‹ CONT ›" ‹ CONT › MENU PROMPT »
```

displays a prompt message, builds a menu with the CONT command assigned to the first menu key, and halts the program for data input. After entering data, pressing CONT resumes program execution. (Note that pressing ← CONT is equivalent to pressing CONT.)

Related Commands: HALT, KILL, PROMPT

CONVERT

Convert Units Command: Converts a source unit object to the dimensions of a target unit.

{}

Level 2	Level 1	→	Level 1
x_1 _*units*$_{source}$	x_2 _*units*$_{target}$	→	x_3 _*units*$_{target}$

Keyboard Access: ◀ UNITS CONV

Affected by Flags: None

Remarks: The source and target units must be compatible. The number part x_2 of the target unit object is ignored.

Related Commands: UBASE, UFACT, →UNIT, UVAL

CORR

Correlation Command: Returns the correlation coefficient of the independent and dependent data columns in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	$x_{correlation}$

Keyboard Access: ◀ STAT FIT CORR

Affected by Flags: None

Remarks: The columns are specified by the first two elements in the reserved variable ΣPAR, set by XCOL and YCOL, respectively. If ΣPAR does not exist, CORR creates it and sets the elements to their default values (1 and 2).

CORR

The correlation is computed with the following formula:

$$\frac{\sum_{i=1}^{n}(x_{in_1}-\overline{x}_{n_1})(x_{in_2}-\overline{x}_{n_2})}{\sqrt{\sum_{i=1}^{n}(x_{in_1}-\overline{x}_{n_1})^2\sum_{i=1}^{n}(x_{in_2}-\overline{x}_{n_2})^2}}$$

where x_{in_1} is the ith coordinate value in column n_1, x_{in_2} is the ith coordinate value in the column n_2, $\overline{x}_{n_1}$ is the mean of the data in column n_1, $\overline{x}_{n_2}$ is the mean of the data in column n_2, and n is the number of data points.

Related Commands: COLΣ, COV, PREDX, PREDY, XCOL, YCOL

COS

Cosine Analytic Function: Returns the cosine of the argument.

{ }

Level 1	→	Level 1
z	→	cos *z*
'*symb*'	→	'COS(*symb*)'
$x_unit_{angular}$	→	cos ($x_unit_{angular}$)

Keyboard Access: COS

Affected by Flags: Numerical Results (−3), Angle Mode (−17, −18)

Remarks: For real arguments, the current angle mode determines the number's interpretation as an angle, unless the angular units are specified.

For complex arguments, $\cos(x + iy) = \cos x\ \cosh y - i\ \sin x\ \sinh y$.

If the argument for COS is a unit object, then the specified angular unit overrides the angle mode to determine the result. Integration and differentiation, on the other hand, always observe the angle mode. Therefore, to correctly integrate or differentiate expressions containing COS with a unit object, the angle mode must be set to Radians (since this is a "neutral" mode).

Related Commands: ACOS, SIN, TAN

COSH

Hyperbolic Cosine Analytic Function: Returns the hyperbolic cosine of the argument.

{ }

Level 1	→	Level 1
z	→	cosh *z*
'*symb*'	→	'COSH(*symb*)'

Keyboard Access: (MTH) HYP COSH

Affected by Flags: Numerical Results (−3)

Remarks: For complex arguments, $\cosh(x + iy) = \cosh x \cos y + i \sinh x \sin y$.

Related Commands: ACOSH, SINH, TANH

COV

Covariance Command: Returns the sample covariance of the independent and dependent data columns in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	$x_{covariance}$

Keyboard Access: (←) (STAT) FIT COV

COV

Affected by Flags: None

Remarks: The columns are specified by the first two elements in reserved variable ΣPAR, set by XCOL and YCOL respectively. If ΣPAR does not exist, COV creates it and sets the elements to their default values (1 and 2).

The covariance is calculated with the following formula:

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_{in_1}-\overline{x_{n_1}})(x_{in_2}-\overline{x_{n_2}})$$

where x_{in_1} is the ith coordinate value in column n_1, x_{in_2} is the ith coordinate value in the column n_2, $\overline{x_{n_1}}$ is the mean of the data in column n_1, $\overline{x_{n_2}}$ is the mean of the data in column n_2, and n is the number of data points.

Related Commands: COLΣ, CORR, PCOV, PREDX, PREDY, XCOL, YCOL

CR

Carriage Right Command: Prints the contents, if any, of the printer buffer.

Keyboard Access: ↰ I/O PRINT CR

Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33)

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

Remarks: When using the HP 82240B Infrared Printer (flag −34 clear), CR leaves the printhead on the right end of the just printed line.

When printing to the serial port (flag −34 set), CR sends to the printer a string that encodes the line termination method. The default termination method is carriage-return/linefeed. The string is the fourth parameter in the reserved variable *PRTPAR*.

Related Commands: DELAY, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PR1

CRDIR

Create Directory Command: Creates an empty subdirectory with the specified name within the current directory.

{ }

Level 1	→	Level 1
'*global*'	→	

Keyboard Access: ◀ MEMORY DIR CRDIR

Affected by Flags: None

Remarks: CRDIR does not change the current directory; evaluate the name of the new subdirectory to make it the current directory.

Related Commands: HOME, PATH, PGDIR, UPDIR

CROSS

Cross Product Command: CROSS returns the cross product $\mathbf{C} = \mathbf{A} \times \mathbf{B}$ of vectors A and B.

{ }

Level 2	Level 1	→	Level 1
[*vector*]$_A$	[*vector*]$_B$	→	[*vector*]$_{A \times B}$

Keyboard Access: MTH VECTR CROSS

Affected by Flags: None

Remarks: The arguments must be vectors having two or three elements, and do not both need the same number of elements. (The HP 48 automatically converts a two-element argument [d_1 d_2] to a three-element argument [d_1 d_2 0].)

CROSS

Related Commands: CNRM, DET, DOT, RNRM

CSWP

Column Swap Command: Swaps columns i and j of the argument matrix and returns the modified matrix, or swaps elements i and j of the argument vector and returns the modified vector.

{ }

Level 3	Level 2	Level 1	→	Level 1
[[*matrix*]]$_1$	n_{columni}	n_{columnj}	→	[[*matrix*]]$_2$
[*vector*]$_1$	n_{elementi}	n_{elementj}	→	[*vector*]$_2$

Keyboard Access: (MTH) MATR COL CSWP

Affected by Flags: None

Remarks: Column numbers are rounded to the nearest integer. Vector arguments are treated as row vectors.

Related Commands: COL+, COL−, RSWP

CYLIN

Cylindrical Mode Command: Sets Cylindrical coordinate mode.

Keyboard Access:

(MODES) ANGL CYLIN

(MTH) VECTR (NXT) CYLIN

Affected by Flags: None

Remarks: CYLIN clears flag −15 and sets flag −16, and displays the R∠Z annunciator.

In Cylindrical mode, vectors are displayed as polar components. Therefore, a 3D vector would appear as $[R \measuredangle \theta \ Z]$.

Related Commands: RECT, SPHERE

C→PX

Complex to Pixel Command: Converts the specified user-unit coordinates to pixel coordinates.

{ }

Level 1	→	Level 1
(x, y)	→	{ #*n* #*m* }

Keyboard Access: PRG FILE NXT C→PX

Affected by Flags: None

Remarks: The user-unit coordinates are derived from the (x_{min}, y_{min}) and (x_{max}, y_{max}) parameters in the reserved variable *PPAR*.

Related Commands: PX→C

C→R

Complex to Real Command: Separates the real and imaginary parts of a complex number or complex array.

{ }

Level 1	→	Level 2	Level 1
(x, y)	→	x	y
[*C-array*]	→	$[\ R\text{-}array\ ]_1$	$[\ R\text{-}array\ ]_2$

C→R

Keyboard Access:

MTH NXT CMPL C→R

PRG TYPE NXT C→R

Affected by Flags: None

Remarks: The result in level 2 represents the real part of the complex argument. The result in level 1 represents the imaginary part of the complex argument.

Related Commands: R→C, RE, IM

DARCY

Darcy Friction Factor Function: Calculates the Darcy friction factor of certain fluid flows.

{ }

Level 2	Level 1	→	Level 1
$x_{e/D}$	y_{Re}	→	x_{Darcy}

Keyboard Access: ↩ EQ LIB UTILS DARCY

Affected by Flags: None

Remarks: DARCY calculates the Fanning friction factor and multiplies it by 4. $x_{e/D}$ is the relative roughness—the ratio of the conduit roughness to its diameter. y_{Re} is the Reynolds number. The function uses different computation routines for laminar flow ($Re \leq 2100$) and turbulent flow ($Re > 2100$). $x_{e/D}$ and y_{Re} must be real numbers or unit objects that reduce to dimensionless numbers, and both numbers must be greater than 0.

Related Commands: FANNING

DATE

Date Command: Returns the system date to level 1.

Level 1	→	Level 1
	→	*date*

Keyboard Access: ◀ TIME DATE

Affected by Flags: Date Format (−42)

Example: If the current date is May 12, 1990, if flag −42 is clear, and if the display mode is Standard, DATE returns 5.12199. (The trailing zeros are dropped.)

Related Commands: DATE+, DDAYS, TIME, TSTR

→DATE

Set Date Command: Sets the system date to *date*.

{ }

Level 1	→	Level 1
date	→	

Keyboard Access: ◀ TIME →DAT

Affected by Flags: Date Format (−42)

Remarks: *date* has the form *MM.DDYYYY* or *DD.MMYYYY*, depending on the state of flag −42. *MM* is month, *DD* is day, and *YYYY* is year. If *YYYY* is not supplied, the current specification for the year is used. The range of allowable dates is January 1, 1991 to December 31, 2090.

→DATE

Example: If flag −42 is set and the current system year is 1995, then 28.07 →DATE sets the system date as July 28, 1995.

Related Commands: →TIME

DATE+

Date Addition Command: Returns a past or future date, given a date in level 2 and a number of days in level 1.

{ }

Level 2	Level 1	→	Level 1
$date_1$	x_{days}	→	$date_{new}$

Keyboard Access: ◀ TIME NXT DATE+

Affected by Flags: Date Format (−42)

Remarks: If x_{days} is negative, DATE+ calculates a past date. The range of allowable dates is October 15, 1582, to December 31, 9999.

Related Commands: DATE, DDAYS

DBUG

Debug Operation: Starts program execution, then suspends it as if HALT were the first program command.

Level 1	→	Level 1
≪ *program* ≫ *or* '*program name*'	→	

Keyboard Access: PRG NXT RUN DBUG

Affected by Flags: None

Remarks: DBUG is not programmable.

Related Commands: HALT, NEXT, SST, SST↓

DDAYS

Delta Days Command: Returns the number of days between two dates.

{}

Level 2	Level 1	→	Level 1
$date_1$	$date_2$	→	x_{days}

Keyboard Access: ◀ TIME NXT DDAYS

Affected by Flags: Date Format (−42)

Remarks: If the level 2 date is chronologically later than the level 1 date, the result is negative. The range of allowable dates is October 15, 1582, to December 31, 9999.

Related Commands: DATE, DATE+

DEC

Decimal Mode Command: Selects decimal base for binary integer operations. (The default base is decimal.)

Keyboard Access: MTH BASE DEC

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: Binary integers require the prefix #. Binary integers entered and returned in decimal base automatically show the suffix d. If the current base is not decimal, then you can enter a decimal

number by ending it with ◻. It will be displayed in the current base when it is entered.

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Related Commands: BIN, HEX, OCT, RCWS, STWS

DECR

Decrement Command: Takes a variable on level 1, subtracts 1, stores the new value back into the original variable, and returns the new value to level 1.

{ }

Level 1	→	Level 1
'*name*'	→	x_{new}

Keyboard Access: ◀ MEMORY ARITH DECR

Affected by Flags: None

Remarks: The contents of *name* must be a real number.

Example: If 35.7 is stored in *A*, 'A' DECR returns 34.7.

The following program counts down from 100 to 0 and leaves the integers 100 to 0 on the stack:

```
«

 100 'A' STO

 WHILE A REPEAT 'A' DECR END

  'A' PURGE

»
```

Related Commands: INCR

DEFINE

Define Variable or Function Command: Stores the expression on the right side of the = in the variable specified on the left side, or creates a user-defined function.

{}

Level 1	→	Level 1
'*name*=*exp*'	→	
'*name*(*name*$_1$ ... *name*$_n$)=*exp*(*name*$_1$ ... *name*$_n$)'	→	

Keyboard Access: ◀ DEF

Affected by Flags: Numerical Results (−3)

For arguments of the form '*name*=*exp*', if flag −3 is set, *expression* will be evaluated to a number before it is stored in *name*. (If *exp* contains a formal variable, DEFINE will error if flag −3 is set.)

Remarks: If the left side of the equation is *name* only, DEFINE stores *exp* in the variable *name*.

If the left side of the equation is *name* followed by parenthetical arguments *name*$_1$ ... *name*$_n$, DEFINE creates a user-defined function and stores it in the variable *name*.

Examples: 'A=2*X' DEFINE stores '2*X' in variable *A*.

'A(X,Y)=2*X+3/Y' DEFINE creates a user-defined function *A*. The contents of *A* is the program « → X Y '2*X+3/Y' ».

Related Commands: STO

DEG

Degrees Command: Sets Degrees angle mode.

Keyboard Access: ↩ MODES ANGL DEG

Affected by Flags: None

Remarks: DEG clears flags −17 and −18, and clears the RAD and GRAD annunciators.

In Degrees angle mode, real-number arguments that represent angles are interpreted as degrees, and real-number results that represent angles are expressed in degrees.

Related Commands: GRAD, RAD

DELALARM

Delete Alarm Command: Deletes the alarm specified in level 1.

{ }

Level 1	→	Level 1
n_{index}	→	

Keyboard Access: ↩ TIME ALRM DELAL

Affected by Flags: None

Remarks: If n_{index} is 0, all alarms in the system alarm list are deleted.

Related Commands: FINDALARM, RCLALARM, STOALARM

DELAY

Delay Command: Specifies how many seconds the HP 48 waits between sending lines of information to the printer.

{ }

Level 1	→	Level 1
x_{delay}	→	

Keyboard Access: ◀ I/O PRINT PRTPR DELAY

Affected by Flags: Printing Device (−34) and I/O Device (−33)

Setting flag −34 directs printer output to the serial port. In this case, flag −33 must be clear.

If flag −34 is set and transmit pacing is enabled (nonzero) in reserved variable *IOPAR*, then XON/XOFF handshaking controls data transmission and the delay setting has no effect.

Remarks: x_{delay} specifies the delay time in seconds. The default delay is 1.8 seconds. The maximum delay is 6.9 seconds. (The sign of x_{delay} is ignored, so −4 DELAY is equivalent to 4 DELAY.)

The delay setting is the first parameter in the reserved variable *PRTPAR*.

A shorter delay setting can be useful when the HP 48 sends multiple lines of information to your printer (for example, when printing a program). To optimize printing efficiency, set the delay just longer than the time the printhead requires to print one line of information.

If you set the delay *shorter* than the time to print one line, you may lose information. Also, as the batteries in the printer lose their charge, the printhead slows down, and, if you have previously decreased the delay, you may have to increase it to avoid losing information. (Battery discharge will not cause the printhead to slow to more than the 1.8-second default delay setting.)

Related Commands: CR, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PR1

DELKEYS

Delete Key Assignments Command: Clears user-defined key assignments.

Level 1	→	Level 1
x_{key}	→	
$\{ x_{key1} \quad \cdots \quad x_{keyn} \}$	→	
0	→	
'S'	→	

Keyboard Access: ↩ MODES KEYS DELK

Affected by Flags: User-Mode Lock (−61) and User Mode (−62) affect the status of the user keyboard.

Remarks: The argument x_{key} is a real number *rc.p* specifying the key by its *r*ow number, its *c*olumn number, and its *p*lane (shift). For a definition of plane, see ASN.

Specifying □ for x_{key} clears *all* user key assignments and restores the standard key assignments.

Specifying 5 as the argument for DELKEYS suppresses all standard key assignments on the user keyboard. This makes keys without user key assignments inactive on the user keyboard. (You can make exceptions using ASN, or restore them all using STOKEYS.) If you are stuck in User mode—probably with a "locked" keyboard—because you have reassigned or suppressed the keys necessary to cancel User mode, do a system halt ("warm start"): press and hold ON and the C key simultaneously, releasing the C key first. This cancels User mode.

Deleted user key assignments still take up from 2.5 to 15 bytes of memory each. You can free this memory by packing your user key assignments by executing RCLKEYS 0 DELKEYS STOKEYS.

Related Commands: ASN, RCLKEYS, STOKEYS

DEPND

Dependent Variable Command: Specifies the dependent variable (and its plotting range for TRUTH plots).

Level 2	Level 1	→	Level 1
	'global'	→	
	{ *global* }	→	
	{ *global* y_{start} y_{end} }	→	
	{ y_{start} y_{end} }	→	
y_{start}	y_{end}	→	

Keyboard Access: ↰ PLOT FFAR DEPN

Affected by Flags: None

Remarks: The specification for the dependent variable name and its plotting range is stored in the reserved variable *PPAR* as follows:

- If the argument is a global variable name, that name replaces the dependent variable entry in *PPAR*.
- If the argument is a list containing a global name, that name replaces the dependent variable name but leaves unchanged any existing plotting range.
- If the argument is a list containing a global name and two real numbers, or a list containing a name, array, and real number, that list replaces the dependent variable entry.
- If the argument is a list containing two real numbers, or two real numbers from levels 1 and 2, those two numbers specify a new plotting range, leaving the dependent variable name unchanged. (LASTARG returns a list, even if the two numbers were entered separately.)

The default entry is *Y*.

The plotting range for the dependent variable is meaningful only for plot type TRUTH, where it restricts the region for which the equation

DEPND

is tested, and for plot type DIFFEQ, where it specifies the initial solution value and absolute error tolerance.

Related Commands: INDEP

DEPTH

Depth Command: Returns a real number representing the number of objects present on the stack (before DEPTH was executed).

Level 1	→	Level 1
	→	*n*

Keyboard Access: ◀ STACK DEPTH

Affected by Flags: None

DET

Determinant Function: Returns the determinant of a square matrix.

{ }

Level 1	→	Level 1
[[*matrix*]]	→	$x_{\text{determinant}}$

Keyboard Access: MTH MATR HOME NXT DET

Affected by Flags: Tiny Element (−54)

Remarks: The argument matrix must be square. DET computes the determinant of 1×1 and 2×2 matrices directly from the defining expression for the determinant. DET computes the determinant of a larger matrix by computing the Crout LU decomposition of the

matrix and accumulating the product of the decomposition's diagonal elements.

Since floating-point division is used to do this, the computed determinant of an integer matrix is often not an integer, even though the actual determinant of an integer matrix must be an integer. DET corrects this by rounding the computed determinant to an integer value. This technique is also used for noninteger matrices with determinants having fewer than 15 nonzero digits: the computed determinant is rounded at the appropriate digit position to restore some or all of the accuracy lost to floating-point arithmetic.

This refinement technique can cause the computed determinant to exhibit discontinuity. To avoid this, you can disable the refinement by setting flag −54.

Example: For a square matrix A, the *minor* of element a_{ij} is the determinant of the submatrix that remains after deleting row i and column j from the original matrix. Given a square matrix in level 3, i in level 2, and j in level 1, the following program, *MINOR* determines the minor of the submatrix:

```
« → M row col

  « M row ROW- DROP col COL- DROP DET

  »

»
```

For example, entering `[[ 1 2 3 ][ 4 5 6 ][ 7 8 9 ]] 2 3 MINOR` returns −6.

Related Commands: CNRM, CROSS, DOT, RNRM

DETACH

Detach Library Command: Detaches the library with the specified number from the current directory. Each library has a unique number. If a port number is specified, it is ignored.

{ }

Level 1	→	Level 1
$n_{library}$	→	
$:n_{port}$ $:n_{library}$	→	

Keyboard Access: ◀ LIBRARY DETHC

Affected by Flags: None

Remarks: A RAM-based library object attached to the HOME directory must be detached before it can be purged, whereas a library attached to any other directory does not. Also, a library object attached to a non-*HOME* directory is *automatically* detached (without using DETACH) whenever a new library object is attached there.

Related Commands: ATTACH, LIBS, PURGE

DIAG→

Array to Matrix Diagonal Command: Takes an array and a specified dimension and returns a matrix whose major diagonal elements are the elements of the array.

Level 2	Level 1	→	Level 1
[*array*]$_{diagonals}$	{ *dim* }	→	[[*matrix*]]

Keyboard Access: MTH MATR NXT DIAG→

Affected by Flags: None

Remarks: Real number dimensions are rounded to integers. If a single dimension is given, a square matrix is returned. If two dimensions are given, the proper order is { *number of rows, number of columns* }. No more than two dimensions can be specified.

If the main diagonal of the resulting matrix has more elements than the array, additional diagonal elements are set to zero. If the main diagonal of the resulting matrix has fewer elements than the array, extra array elements are dropped.

Related Commands: →DIAG

→DIAG

Matrix Diagonal to Array Command: Returns a vector that contains the major diagonal elements of a matrix.

{ }

Level 1	→	Level 1
[[*matrix*]]	→	[*vector*]$_{\text{diagonals}}$

Keyboard Access: (MTH) MATR (NXT) →DIAG

Affected by Flags: None

Remarks: The input matrix does not have to be square.

Related Commands: DIAG→

DIFFEQ

Differential Equation Plot Type Command: Sets the plot type to DIFFEQ.

Keyboard Access: ◀ PLOT PTYPE DIFFE

Affected by Flags: None

Remarks: When the plot type is DIFFEQ and the reserved variable *EQ* does not contain a list, the initial value problem is solved and plotted over an interval using the Runge-Kutta-Fehlberg (4,5) method. The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

$$\{\ (x_{\mathtt{min}},\ y_{\mathtt{min}})\ (x_{\mathtt{max}},\ y_{\mathtt{max}})\ \mathit{indep}\ \mathit{res}\ \mathit{axes}\ \mathit{ptype}\ \mathit{depend}\ \}$$

For plot type DIFFEQ, the elements of *PPAR* are used as follows:

- $(x_{\mathsf{min}}, y_{\mathsf{min}})$ is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (-6.5,-3.1).
- $(x_{\mathsf{max}}, y_{\mathsf{max}})$ is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a list, { 'X' x_0 x_f }, containing a name that specifies the independent variable, and two numbers that specify the initial and final values for the independent variable. The default values for these elements are { 'X' 0 x_{max} }.
- *res* is a real number specifying the maximum interval, in user-unit coordinates, between values of the independent variable. The default value is 0. If *res* does not equal zero, then the maximum interval is *res*. If *res* equals zero, the maximum interval is unlimited.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. If the solution is real-valued, these strings can specify the dependent or the independent variable; if the solution is vector valued, the strings can specify a solution component:
  - "0" specifies the independent variable (X)

- □ "1" specifies the dependent variable (Y)
- □ "n" specifies a solution component Y_n

If *axes* contains any strings other than "0", "1", or "n", the DIFFEQ-plotter uses the default strings "0" and "1", and plots the independent variable on the horizontal axis and the dependent variable on the vertical.

- *ptype* is a command name specifying the plot type. Executing the command DIFFEQ places the command name DIFFEQ in *PPAR*.
- *depend* is a list, { 'Y' y_0 x_{ErrTol} }, containing a name that specifies the dependent variable (the solution), and two numbers that specify the inital value of Y and the global absolute error tolerance in the solution Y. The default values for these elements are { 'Y' 0 .0001 }.

EQ must define the right-hand side of the initial value problem Y'(X)=F(X,Y). Y can return a real value or real vector when evaluated.

The DIFFEQ-plotter attempts to make the interval between values of the independent variable as large as possible, while keeping the computed solution within the specified error tolerance x_{ErrTol}. This tolerance may hold only at the computed points. Straight lines are drawn between computed step endpoints, and these lines may not accurately represent the actual shape of the solution. *res* limits the maximum interval size to provide higher plot resolution.

On exit from DIFFEQ plot, the first elements of *indep* and *depnd* (identifiers) contain the final values of X and Y, respectively.

If *EQ* contains a list, the inital value problem is solved and plotted using a combination of Rosenbrock (3,4) and Runge-Kutta-Fehlberg (4,5) methods. In this case DIFFEQ uses RRKSTEP to calculate y_f, and *EQ* must contain two additional elements:

- The second element of *EQ* must evaluate to the partial derivative of Y' with respect to X, and can return a real value or real vector when evaluated.
- The third element of *EQ* must evaluate to the partial derivative of Y' with respect to Y, and can return a real value or a real matrix when evaluated.

DIFFEQ

Related Commands: AXES, CONIC, FUNCTION, PARAMETRIC, POLAR, RKFSTEP, RRKSTEP, TRUTH

DISP

Display Command: Displays *obj* in the *n*th display line.

{ }

Level 2	Level 1	→	Level 1
obj	*n*	→	

Keyboard Access: PRG NXT OUT DISP

Affected by Flags: None

Remarks: $n \leq 1$ indicates the top line of the display; $n \geq 7$ indicates the bottom line.

To facilitate the display of messages, strings are displayed without the surrounding " " delimiters. All other objects are displayed in the same form as would be used if the object were in level 1 in the multiline display format. If the object display requires more than one display line, the display starts in line *n*, and continues down the display either to the end of the object or the bottom of the display.

The object displayed by DISP persists in the display only until the keyboard is ready for input. The FREEZE command can be used to cause the object to persist in the display until a key is pressed.

Example: The program

```
« "ENTER Data Now" 1 DISP 7 FREEZE HALT »
```

displays ENTER Data Now at the top of the display, "freezes" the entire display, and halts.

Related Commands: FREEZE, HALT, INPUT, PROMPT

DO

DO Indefinite Loop Structure Command: Starts DO ... UNTIL ... END indefinite loop structure.

Level 1	→	Level 1
DO	→	
UNTIL	→	
END *T/F*	→	

Keyboard Access: PRG BRCH DO DO

Affected by Flags: None

Remarks: DO ... UNTIL ... END executes a loop repeatedly until a test returns a true (nonzero) result. Since the test clause is executed after the loop clause, the loop is always executed at least once. The syntax is:

DO *loop-clause* UNTIL *test-clause* END

DO starts execution of the loop clause. UNTIL ends the loop clause and begins the test clause. The test clause must return a test result to the stack. END removes the test result from the stack. If its value is zero, the loop clause is executed again; otherwise, execution resumes following END.

Example: The following program counts down from 100 to 0 and leaves the integers 100 to 0 on the stack:

```
«
  100 'A' STO A
    DO 'A' DECR
    UNTIL 'A==0'
    END
  'A' PURGE
»
```

DO

Related Commands: END, UNTIL, WHILE

DOERR

Do Error Command: Executes a "user-specified" error, causing a program to behave exactly as if a normal error had occurred during program execution.

{ }

Level 1	→	Level 1
n_{error}	→	
$\#n_{error}$	→	
"*error*"	→	
0	→	

Keyboard Access: PRG NXT ERROR DOERR

Affected by Flags: None

Remarks: DOERR causes a program to behave exactly as if a normal error has occurred during program execution. The error message depends on the argument provided to DOERR:

- n_{error} or $\#n_{error}$ display the corresponding built-in error message.
- "*error*" displays the contents of the string. (A subsequent execution of ERRM returns "*error*". ERRN returns # 70000h.)
- 0 abandons program execution without displaying a message—0 DOERR is equivalent to pressing CANCEL.

Example: The following program takes a number from the stack and returns an error if the number is greater than 10:

```
« → X
  CASE
  'X>10'
  THEN "X IS TOO BIG" DOERR END
```

```
  END
»
```

Related Commands: ERRM, ERRN, ERR0

DOLIST

Do to List Command: Applies commands, programs, or user-defined functions to lists.

Level n...Level 3	Level 2	Level 1	→	Level 1
{ *list* }$_1$ ... { *list* }$_n$	*n*	≪ *program* ≫	→	{ *results* }
{ *list* }$_1$ ... { *list* }$_n$	*n*	*command*	→	{ *results* }
{ *list* }$_1$ ... { *list* }$_n$	*n*	*name*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_n$	≪ *program* ≫	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_n$	*command*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_n$	*name*	→	{ *results* }

Keyboard Access: PRG LIST FROC DOLIS

Affected by Flags: None

Remarks: The number of lists, n, can be omitted when the level 1 argument is any of the following:

- A command.
- A program containing exactly one command (e.g. ≪ DUP ≫).
- A program conforming to the structure of a user-defined function.

The level 1 object can be a local or global name that refers to a program or command.

All lists must be the same length l. The program is executed l times: on the ith iteration, n objects each taken from the ith position in each list are entered on the stack in the same order as in their original lists, and the program is executed. The results from each execution are left on the stack. After the final iteration, any new results are combined into a single list.

DOLIST

Example: { 1 2 3 }{ 4 5 6 }{ 7 8 9 } 3 « + » DOLIST returns { 11 26 45 }.

Related Commands: DOSUBS, ENDSUB, NSUB, STREAM

DOSUBS

Do to Sublist Command: Applies a program or command to groups of elements in a list.

Level 3	Level 2	Level 1	→	Level 1
$\{ list \}_1$	n	$\ll program \gg$	→	$\{ list \}_2$
$\{ list \}_1$	n	*command*	→	$\{ list \}_2$
$\{ list \}_1$	n	*name*	→	$\{ list \}_2$
	$\{ list \}_1$	$\ll program \gg$	→	$\{ list \}_2$
	$\{ list \}_1$	*command*	→	$\{ list \}_2$
	$\{ list \}_1$	*name*	→	$\{ list \}_2$

Keyboard Access: PRG LIST PROC DOSUB

Affected by Flags: None

Remarks: The real number n can be omitted when the level 1 argument is one of the following:

- A command.
- A user program containing a single command.
- A program with a user-defined function structure.
- A global or local name that refers to one of the above.

The first iteration uses elements 1 through n from the list; the second iteration uses elements 2 through $n+1$; and so on. In general, the m^{th} iteration uses the elements from the list corresponding to positions m through $m+n-1$.

During an iteration, the position of the first element used in that iteration is available to the user using the command NSUB, and the number of groups of elements is available using the command

ENDSUB. Both of these commands return an Undefined Local Name error if executed when DOSUBS is not active.

DOSUBS returns the Invalid User Function error if the level 1 argument is a user program that does not contain only one command and does not have a user-defined function structure. DOSUBS also returns the Wrong Argument Count error if the level 1 argument is a command that does not accept 1 to 5 arguments of specific types (DUP, ROT, or →LIST, for example).

Examples: { A B C D E } « - » DOSUBS returns
{ 'A-B' 'B-C' 'C-D' 'D-E' }.

{ A B C } 2 « DUP * * » DOSUBS returns
{ 'A*(B*B)' 'B*(C*C)' }.

Entering

```
{ 1 2 3 4 5 }
« → a b

  « CASE  'NSUB==1' THEN a END

    'NSUB==ENDSUB' THEN b END

    'a+b' EVAL

   END

  »

»

DOSUBS
```

returns
{ 1 5 7 5 }.

Related Commands: DOLIST, ENDSUB, NSUB, STREAM

DOT

Dot Product Command: Returns the dot product $\mathbf{A}\cdot\mathbf{B}$ of two arrays **A** and **B**, calculated as the sum of the products of the corresponding elements of the two arrays.

{ }

Level 2	Level 1	→	Level 1
[*array A*]	[*array B*]	→	*x*

Keyboard Access: (MTH) VECTR DOT

Affected by Flags: None

Remarks: Both arrays must have the same dimensions.

Some authorities define the dot product of two complex arrays as the sum of the products of the conjugated elements of one array with their corresponding elements from the other array. The HP 48 uses the ordinary products without conjugation. If you prefer the alternate definition, apply CONJ to one or both arrays before using DOT.

Example: `[ 1 2 3 ] [ 4 5 6 ] DOT` returns `32` (by calculating $1 \times 4 + 2 \times 5 + 3 \times 6$).

Related Commands: CNRM, CROSS, DET, RNRM

DRAW

Draw Plot Command: Plots the mathematical data in the reserved variable EQ or the statistical data in the reserved variable ΣDAT, using the specified x- and y-axis display ranges.

Keyboard Access: ← (PLOT) DRAW

Affected by Flags: Simultaneous or Sequential Plot (−28), Curve Filling (−31)

Remarks: The plot type determines if the data in the reserved variable EQ or the data in the reserved variable ΣDAT is plotted.

DRAW does not erase *PICT* before plotting—execute ERASE to do so. DRAW does not draw axes—execute DRAX to do so.

When DRAW is executed from a program, the graphics display, which shows the resultant plot, does not persist unless PICTURE, PVIEW (with an empty list argument), or FREEZE is subsequently executed.

Related Commands: AUTO, AXES, DRAX, ERASE, FREEZE, PICTURE, LABEL, PVIEW

DRAX

Draw Axes Command: Draws axes in *PICT*.

Keyboard Access: ◀ PLOT DRAX

Affected by Flags: None

Remarks: The coordinates of the axes intersection are specified by AXES. Axes tick-marks are specified in PPAR with the ATICK, or AXES command. DRAX does not draw axes labels—execute LABEL to do so.

Related Commands: AXES, DRAW, LABEL

DROP

Drop Object Command: Removes the level 1 object from the stack.

Level 1	→	Level 1
obj	→	

Keyboard Access: ◀ DROP

Affected by Flags: None

Related Commands: CLEAR, DROPN, DROP2

DROPN

Drop n Objects Command Removes the first $n + 1$ objects from the stack (the first n objects excluding the integer n itself).

Level n+1 ... Level 2	Level 1	→	Level 1
obj_1 ... obj_n	n	→	

Keyboard Access: ↰ STACK NXT DROPN

Affected by Flags: None

Related Commands: CLEAR, DROP, DROP2

DROP2

Drop 2 Objects Command: Removes the first two objects from the stack.

Level 2	Level 1	→	Level 1
obj_1	obj_2	→	

Keyboard Access: ↰ STACK NXT DROP2

Affected by Flags: None

Related Commands: CLEAR, DROP, DROPN

DTAG

Delete Tag Command: DTAG removes all tags (labels) from an object.

{}

Level 1	→	Level 1
:tag:obj	→	*obj*

Keyboard Access: PRG TYPE NXT DTAG

Affected by Flags: None

Remarks: The leading colon is not shown for readability when the tagged object is on the stack.

DTAG has no effect on an untagged object.

Related Commands: LIST→, →TAG

DUP

Duplicate Object Command: DUP returns a copy to level 1 of the object in level 1.

Level 1	→	Level 2	Level 1
obj	→	*obj*	*obj*

Keyboard Access:

Pressing ENTER duplicates the item on level 1.

◀ STACK NXT DUP

Affected by Flags: None

Related Commands: DUPN, DUP2, PICK

DUPN

Duplicate n Objects Command: Takes an integer n from level 1 of the stack, and returns copies of the objects in stack levels $2n$ through $n + 1$.

Lvl n+1...Lvl 2	Lvl 1	→	Lvl 2n...Lvl n+1	Lvl n...Lvl 1
$obj_n \ldots obj_1$	n	→	$obj_n \ldots obj_1$	$obj_n \ldots obj_1$

Keyboard Access: ← STACK NXT DUPN

Affected by Flags: None

Related Commands: DUP, DUP2, PICK

DUP2

Duplicate 2 Objects Command: DUP2 returns copies of the objects in levels 1 and 2 of the stack.

Level 2	Level 1	→	Level 4	Level 3	Level 2	Level 1
obj_2	obj_1	→	obj_2	obj_1	obj_2	obj_1

Keyboard Access: ← STACK NXT DUP2

Affected by Flags: None

Related Commands: DUP, DUPN, PICK

D→R

Degrees to Radians Function: Converts a real number representing an angle in degrees to its equivalent in radians.

{ }

Level 1	→	Level 1
x	→	(π/180) *x*
'*symb*'	→	'D→R(*symb*)'

Keyboard Access: MTH REAL NXT NXT D→R

Affected by Flags: Numerical Results (−3)

Remarks: This function operates independently of the angle mode.

Related Commands R→D

e

e Function: Returns the symbolic constant *e* or its numerical representation, 2.71828182846.

Level 1	→	Level 1
	→	'e'
	→	2.71828182846

Keyboard Access:

α ◀ E ENTER

MTH NXT CONS E

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

e

When evaluated, *e* returns its numerical representation if flag −2 or flag −3 is set; otherwise, *e* returns its symbolic representation.

Remarks: The number returned for e is the closest approximation of the constant *e* to 12-digit accuracy. For exponentiation, use the expression 'EXP(X)' rather than 'e^X', since the function EXP uses a special algorithm to compute the exponential to greater accuracy.

Related Commands: EXP, EXPM, i, LN, LNP1, MAXR, MINR, π

EGV

Eigenvalues and Eigenvectors Command: Computes the eigenvalues and right eigenvectors for a square matrix.

{ }

Level 1	→	Level 2	Level 1
[[*matrix*]]$_A$	→	[[*matrix*]]$_{EVec}$	[*vector*]$_{EVal}$

Keyboard Access: MTH MATR NXT EGV

Affected by Flags: None

Remarks: The resulting vector EVal contains the computed eigenvalues. The columns of matrix EVec contain the right eigenvectors corresponding to the elements of vector EVal.

The computed results should minimize (within computational precision):

$$\frac{|A \cdot EVec - EVec \cdot \mathrm{diag}(EVal)|}{n \cdot |A|}$$

where diag($EVal$) denotes the $n \times n$ diagonal matrix containing the eigenvalues $EVal$.

Related Commands: EGVL

EGVL

Eigenvalues Command: Computes the eigenvalues of a square matrix.

{ }

Level 1	→	Level 1
[[*matrix*]]$_A$	→	[*vector*]$_{EVal}$

Keyboard Access: (MTH) MATR (NXT) EGVL

Affected by Flags: None

Remarks: The resulting vector L contains the computed eigenvalues.

Related Commands: EGV

ELSE

ELSE Command: Starts false clause in conditional or error-trapping structure.

See the IF and IFERR keyword entries for syntax information.

Keyboard Access:

(PRG) BRCH IF ELSE

(PRG) (NXT) ERROR IFERR ELSE

Remarks: See the IF and IFERR keyword entries for more information.

Related Commands: IF, IFERR, THEN, END

END

END Command: Ends conditional, error-trapping, and indefinite loop structures.

See the IF, CASE, IFERR, DO, and WHILE keyword entries for syntax information.

Keyboard Access:

PRG BRCH IF END

PRG BRCH CASE END

PRG BRCH DO END

PRG BRCH WHILE END

PRG NXT ERROR IFERR END

Remarks: See the IF, CASE, IFERR, DO, and WHILE keyword entries for more information.

Related Commands: IF, CASE, DO, ELSE, IFERR, REPEAT, THEN, UNTIL, WHILE,

ENDSUB

Ending Sublist Command: Provides a way to access the total number of sublists contained in the list used by DOSUBS.

Keyboard Access: PRG LIST PROC ENDS

Affected by Flags: None

Remarks: Returns an Undefined Local Name error if executed when DOSUBS is not active.

Example: The following program subtracts the number of elements in a list from each element in the list:

```
« → a
  « a 1
    « ENDSUB -
```

```
    »
  » DOSUBS
»
```

Related Commands: DOSUBS, NSUB

ENG

Engineering Mode Command: Sets the number display format to Engineering mode, which displays one to three digits to the left of the fraction mark (decimal point) and an exponent that is a multiple of three. The total number of significant digits displayed is $n + 1$.

{ }

Level 1	→	Level 1
n	→	

Keyboard Access: ◀ MODES FMT ENG

Affected by Flags: None

Remarks: Engineering mode uses $n + 1$ significant digits, where $0 \leq n \leq 11$. (Values for n outside this range are rounded up or down.) A number is displayed or printed as follows:

(sign) mantissa E *(sign) exponent*

where the mantissa is of the form $(nn)n.(n \ldots)$ (with up to 12 digits total) and the exponent has one to three digits.

A number with an exponent of −499 is displayed automatically in Scientific mode.

Example: The number 103.6 in Engineering mode with five significant digits (n=4) would appear as `103.60E0`. This same number with one significant digit (n=0) would appear as `100.E0`.

Related Commands: FIX, SCI, STD

EQ→

Equation to Stack Command: Separates an equation into its left and right sides.

{}

Level 1	→	Level 2	Level 1
'$symb_1 = symb_2$'	→	'$symb_1$'	'$symb_2$'
z	→	z	0
'*name*'	→	'*name*'	0
x_unit	→	*x_unit*	0
'*symb*'	→	'*symb*'	0

Keyboard Access: PRG TYPE NXT EQ→

Affected by Flags: None

Remarks: If the argument is an expression, it is treated as an equation whose right side equals zero.

Related Commands: ARRY→, DTAG, LIST→, OBJ→, STR→

EQNLIB

Equation Library Command: Starts the Equation Library application.

Keyboard Access: ← EQ LIB EQLIB EQNLI

Affected by Flags: None

Remarks: The Equation Library is a collection of equations and commands useful for solving typical science and engineering problems.

Related Commands: MSOLVR, SOLVEQN

ERASE

Erase PICT Command: Erases *PICT*, leaving a blank *PICT* of the same dimensions.

Keyboard Access:

◀ PICTURE EDIT NXT ERASE

◀ PICTURE ◀ CLEAR

◀ PLOT ERASE

Affected by Flags: None

Related Commands: DRAW

ERRM

Error Message Command: Returns a string containing the error message of the most recent calculator error.

Level 1	→	Level 1
	→	"*error message*"

Keyboard Access: PRG NXT ERROR ERRM

Affected by Flags: None

Remarks: ERRM returns the string for an error generated by DOERR. If the argument to DOERR was 0, the string returned by ERRM is empty.

Example: The program « IFERR + THEN ERRM END » returns "Bad Argument Type" to level 1 if improper arguments (for example, a complex number and a binary integer) are in levels 1 and 2.

Related Commands: DOERR, ERRN, ERR0

ERRN

Error Number Command: Returns the error number of the most recent calculator error.

Level 1	→	Level 1
	→	#n_{error}

Keyboard Access: PRG NXT ERROR ERRN

Affected by Flags: None

Remarks: If the most recent error was generated by DOERR with a string argument, ERRN returns # 70000h. If the most recent error was generated by DOERR with a binary integer argument, ERRN returns that binary integer. (If the most recent error was generated by DOERR with a real number argument, ERRN returns the binary integer conversion of the real number.)

Example: The program « IFERR + THEN ERRN END » returns # 202h to level 1 if improper arguments (for, example, a complex number and a binary integer) are in levels 1 and 2.

Related Commands: DOERR, ERRM, ERR0

ERR0

Clear Last Error Number Command: Clears the last error number so that a subsequent execution of ERRN returns # 0h, and clears the last error message.

Keyboard Access: PRG NXT ERROR ERR0

Affected by Flags: None

Related Commands: DOERR, ERRM, ERRN

EVAL

Evaluate Object Command: Evaluates the object.

Level 1	→	Level 1
obj	→	(*see below*)

Keyboard Access: EVAL

Affected by Flags: Numerical Results (−3)

Remarks: The following table describes the effect of the evaluation on different object types.

Obj. Type	Effect of Evaluation
Local Name	Recalls the contents of the variable.
Global Name	*Calls* the contents of the variable: ■ A name is evaluated. ■ A program is evaluated. ■ A directory becomes the current directory. ■ Other objects are put on the stack. If no variable exists for a given name, evaluating the name returns the name to the stack.
Program	*Enters* each object in the program: ■ Names are evaluated (unless quoted). ■ Commands are evaluated. ■ Other objects are put on the stack.
List	*Enters* each object in the list: ■ Names are evaluated. ■ Commands are evaluated. ■ Programs are evaluated. ■ Other objects are put on the stack.

EVAL

Obj. Type	Effect of Evaluation
Tagged	If the tag specifies a port, recalls and evaluates the specified object. Otherwise, puts the untagged object on the stack.
Algebraic	*Enters* each object in the algebraic expression: ■ Names are evaluated. ■ Commands are evaluated. ■ Other objects are put on the stack.
Command, Function, XLIB Name	Evaluates the specified object.
Other Objects	Puts the object on the stack.

To evaluate a symbolic argument to a numerical result, evaluate the argument in Numerical Result mode (flag −3 set) or execute →NUM on that function.

Related Commands: →NUM, SYSEVAL

EXP

Exponential Analytic Function: Returns the exponential, or natural antilogarithm, of the argument; that is, e raised to the given power.

{ }

Level 1	→	Level 1
z	→	e^z
'*symb*'	→	'EXP(*symb*)'

Keyboard Access: ◀ e^x

Affected by Flags: Numerical Results (−3)

Remarks: EXP uses extended precision constants and a special algorithm to compute its result to full 12-digit precision for all arguments that do not trigger and underflow or overflow error.

EXP provides a more accurate result for the exponential than can be obtained by using e $\boxed{y^x}$. The difference in accuracy increases as z increases. For example:

z	**EXP**(z)	e^z
3	20.0855369232	20.0855369232
10	22026.4657948	22026.4657949
100	2.68811714182E43	2.68811714191E43
500	1.40359221785E217	1.40359221809E217
1000	1.97007111402E434	1.97007111469E434

For complex arguments,

$$e^{(x,y)} = e^x cosy + ie^x siny$$

Related Commands: ALOG, EXPM, LN, LOG

EXPAN

Expand Products Command: Rewrites an algebraic expression or equation by expanding products and powers.

{ }

Level 1	→	**Level 1**
'$symb_1$'	→	'$symb_2$'
x	→	x
(x, y)	→	(x, y)

Keyboard Access: ◀ SYMBOLIC EXPA

Affected by Flags: None

Examples: 'A*(B+C)' EXPAN returns 'A*B+A*C'.

EXPAN

'A^(B+C)' EXPAN returns 'A^B*A^C'.

'X^5' EXPAN returns 'X*X^4'.

'(X+Y)^2' EXPAN returns 'X^2+2*X*Y+Y^2'.

Related Commands: COLCT, ISOL, QUAD, SHOW

EXPFIT

Exponential Curve Fit Command: Stores EXPFIT as the fifth parameter in the reserved variable ΣPAR, indicating that subsequent executions of LR are to use the exponential curve fitting model.

Keyboard Access: ↶ STAT ΣPAR MODL EXPFI

Affected by Flags: None

Remarks: LINFIT is the default specification in ΣPAR. For a description of ΣPAR, see appendix D, "Reserved Variables."

Related Commands: BESTFIT, LR, LINFIT, LOGFIT, PWRFIT

EXPM

Exponential Minus 1 Analytic Function: Returns $e^x - 1$.

{ }

Level 1	→	Level 1
x	→	$e^x - 1$
'*symb*'	→	'EXPM(*symb*)'

Keyboard Access: MTH HYP NXT EXPM

Affected by Flags: Numerical Results (−3)

Remarks: For values of x close to zero, 'EXPM(x)' returns a more accurate result than does 'EXP(x)−1'. (Using EXPM allows

both the argument and the result to be near zero, and avoids an intermediate result near 1. The calculator can express numbers within 10^{-449} of zero, but within only 10^{-11} of 1.)

Related Commands: EXP, LNP1

EYEPT

Eye Point Command: Specifies the coordinates of the eye point in a perspective plot.

{ }

Level 3	Level 2	Level 1	→	Level 1
x_{point}	y_{point}	z_{point}	→	

Keyboard Access: ◀ PLOT NXT 3D VPAR NXT EYEPT

Affected by Flags: None

Remarks: x_{point}, y_{point}, and z_{point} are real numbers that set the x-, y-, and z-coordinates as the eye-point from which to view a 3D plot's view volume. The y-coordinate must always be 1 unit less than the view volume's nearest point (y_{near} of YVOL). These coordinates are stored in the reserved variable *VPAR*.

Related Commands: NUMX, NUMY, XVOL, XXRNG, YVOL, YYRNG, ZVOL

F0λ

Black Body Emissive Power Function: Returns the fraction of total black-body emissive power.

{ }

Level 2	Level 1	→	Level 1
y_{lambda}	x_T	→	x_{power}
y_{lambda}	'*symb*'	→	'F0λ(y_{lambda},*symb*)'
'*symb*'	x_T	→	'F0λ(*symb*,x_T)'
'$symb_1$'	'$symb_2$'	→	'F0λ($symb_1$,$symb_2$)'

Keyboard Access: ◀ EQ LIB UTILS F0λ

Affected by Flags: Numerical Results (−3)

Remarks: F0λ calculates the fraction of total black-body emissive power at temperature x_T between wavelengths 0 and y_{lambda}. If units are not specified, y_{lambda} has implied units of meters and x_T has implied units of K.

F0λ returns a dimensionless fraction.

FACT

Factorial (Gamma) Function: Provided for compatibility with the HP 28. FACT is the same as !. See !.

{ }

Level 1	→	Level 1
n	→	*n*!
x	→	$\Gamma(x+1)$
'*symb*'	→	'(symb)!'

Keyboard Access: None. Must be typed in.

FANNING

Fanning Friction Factor Function: Calculates the Fanning friction factor of certain fluid flows.

{ }

Level 2	Level 1	→	Level 1
$x_{x/D}$	y_{Re}	→	$x_{fanning}$
$x_{x/D}$	'*symb*'	→	'FANNING($x_{x/D}$,*symb*)'
'*symb*'	y_{Re}	→	'FANNING(*symb*,y_{Re})'
'$symb_1$'	'$symb_2$'	→	'FANNING($symb_1$,$symb_2$)'

Keyboard Access: ◀ EQ LIB UTILS FANNI

Affected by Flags: Numerical Results (−3)

Remarks: FANNING calculates the Fanning friction factor, a correction factor for the frictional effects of fluid flows having constant temperature, cross-section, velocity, and viscosity (a typical pipe flow, for example). $x_{x/D}$ is the relative roughness (the ratio of the conduit roughness to its diameter). y_{Re} is the Reynolds number. The function uses different computation routines for laminar flow ($Re \leq 2100$) and turbulent flow ($Re > 2100$). $x_{x/D}$ and y_{Re} must be real numbers or unit objects that reduce to dimensionless numbers, and both numbers must be greater than 0.

Related Commands: DARCY

FC?

Flag Clear? Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is clear, and returns a corresponding test result: 1 (true) if the flag is clear or 0 (false) if the flag is set.

{ }

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	0/1

Keyboard Access:

PRG TEST NXT NXT FC?

◀ MODES FLAG FC?

Affected by Flags: None

Related Commands: CF, FC?C, FS?, FS?C, SF

FC?C

Flag Clear? Clear Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is clear, and returns a corresponding test result: 1 (true) if the flag is clear or 0 (false) if the flag is set. After testing, clears the flag.

{ }

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	0/1

Keyboard Access:

PRG TEST NXT NXT FC?C

◀ MODES FLAG FC?C

Affected by Flags: None

Example: If flag −44 is set, −44 FC?C returns 0 to level 1 and clears flag −44.

Related Commands: CF, FC?, FS?, FS?C, SF

FFT

Discrete Fourier Transform Command: Computes the one- or two-dimensional discrete Fourier transform of an array.

{ }

Level 1	→	Level 1
[*array*]$_1$	→	[*array*]$_2$

Keyboard Access: (MTH) (NXT) FFT FFT

Affected by Flags: None

Remarks: If the argument is an N-vector or an $N \times 1$ or $1 \times N$ matrix, FFT computes the one-dimensional transform. If the argument is an $M \times N$ matrix, FFT computes the two-dimensional transform. M and N must be integral powers of 2.

The one-dimensional discrete Fourier transform of an N-vector X is the N-vector Y where:

$$Y_k = \sum_{n=0}^{N-1} X_n e^{-\frac{2\pi ikn}{N}}, \qquad i = \sqrt{-1}$$

for $k = 0,\ 1,\ ...,\ N-1$.

The two-dimensional discrete Fourier transform of an $M \times N$ matrix X is the $M \times N$ matrix Y where:

$$Y_{kl} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{mn} e^{-\frac{2\pi ikm}{M}} e^{-\frac{2\pi iln}{N}}, \qquad i = \sqrt{-1}$$

for $k = 0,\ 1,\ ...,\ M-1$ and $l = 0,\ 1,\ ...,\ N-1$.

FFT

The discrete Fourier transform and its inverse are defined for any positive sequence length. However, the calculation can be performed very rapidly when the sequence length is a power of two, and the resulting algorithms are called the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT).

The FFT command uses truncated 15-digit arithmetic and intermediate storage, then rounds the result to 12-digit precision.

Related Commands: IFFT

FINDALARM

Find Alarm Command: Returns the alarm index n_{index} of the first alarm due after the specified time.

Level 1	→	Level 1
date	→	n_{index}
{ *date time* }	→	n_{index}
0	→	n_{index}

Keyboard Access: ↩ TIME ALRM FINDM

Affected by Flags: Date Format (−42)

Remarks: If the level 1 argument is a real number *date*, FINDALARM returns the index of the first alarm due after 12:00 AM on that date. If the argument is a list { *date time* }, it returns the index of the first alarm due after that date and time. If the argument is the real number 0, FINDALARM returns the first *past-due* alarm. For any of the three arguments, FINDALARM returns 0 if no alarm is found.

Related Commands: DELALARM, RCLALARM, STOALARM

FINISH

Finish Server Mode Command: Terminates Kermit Server mode in a device connected to an HP 48

Keyboard Access: ◀ I/O SRVR FINI

Affected by Flags: I/O Device flag (−33), I/O Messages (−39)

Remarks: FINISH is used by a local Kermit device to tell a server Kermit (connected via the serial port or the IR port) to exit Server mode.

Related Commands: BAUD, CKSM, KGET, PARITY, PKT, RECN, RECV, SEND, SERVER

FIX

Fix Mode Command: Sets the number display format to Fix mode, which rounds the display to n decimal places.

{ }

Level 1	→	Level 1
n	→	

Keyboard Access: ◀ MODES FMT FIX

Affected by Flags: None

Remarks: Fix mode shows n digits to the right of the fraction mark (decimal point), where $0 \leq n \leq 11$. (Values for n outside this range are rounded to the nearest integer.) A number is displayed or printed as follows:

(sign) mantissa

where the mantissa can be of any form. However, the calculator automatically displays a number in Scientific mode if either of the following is true:

FIX

- The number of digits to be displayed exceeds 12.
- A nonzero value rounded to n decimal places otherwise would be displayed as zero.

Example: The number 103.6 in Fix mode to four decimal places would appear as `103.6000`.

Related Commands: FIX, SCI, STD

FLOOR

Floor Function: Returns the greatest integer that is less than or equal to the argument.

{ }

Level 1	→	Level 1
x	→	*n*
x_unit	→	*n_unit*
'*symb*'	→	'FLOOR(*symb*)'

Keyboard Access: MTH REAL NXT NXT FLOOR

Affected by Flags: Numerical Results (−3)

Examples: `3.2 FLOOR` returns `3`; `-3.2 FLOOR` returns `-4`.

Related Commands: CEIL, IP, RND, TRNC

FOR

FOR Definite Loop Structure Command: Starts FOR ... NEXT and FOR ... STEP definite loop structures.

	Level 2	Level 1	→	Level 1
FOR	x_{start}	x_{finish}	→	
NEXT			→	
FOR	x_{start}	x_{finish}	→	
STEP		$x_{increment}$	→	
STEP		$'symb_{increment}'$	→	

Keyboard Access:
To begin a definite loop:

PRG BRCH FOR FOR

To type FOR NEXT

PRG BRCH ↩ FOR

To type FOR STEP:

PRG BRCH ↪ FOR

Affected by Flags: None

Remarks: Definite loop structures execute a command or sequence of commands a specified number of times.

- A FOR ... NEXT loop executes a program segment a specified number of times using a local variable as the loop counter. You can use this variable within the loop. The syntax is this:

  x_{start} x_{finish} FOR *counter loop-clause* NEXT

  FOR takes x_{start} and x_{finish} from the stack as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Then, the loop clause is executed; *counter* can be referenced or have its value changed within the loop clause. NEXT increments *counter* by one, and then tests whether

counter is less than or equal to x_{finish}. If so, the loop clause is repeated (with the new value of *counter*).

When the loop is exited, *counter* is purged.

- FOR ... STEP works just like FOR ... NEXT, except that it lets you specify an increment value other than 1. The syntax is:

x_start x_finish FOR *counter loop-clause x_increment* STEP

FOR takes x_{start} and x_{finish} from the stack as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Next, the loop clause is executed; *counter* can can be referenced or have its value changed within the loop clause. STEP takes $x_{\mathsf{increment}}$ from the stack and increments *counter* by that value. If the argument of STEP is an algebraic expression or a name, it is automatically evaluated to a number.

The increment value can be positive or negative. If the increment is positive, the loop is executed again when *counter* is less than or equal to x_{finish}. If the increment is negative, the loop is executed when *counter* is greater than or equal to x_{finish}.

When the loop is exited, *counter* is purged.

Example: The following program sums all odd integers in the range 1 to 100:

```
« 0 1 100
  FOR I I + 2 STEP
»
```

Related Commands: NEXT, START, STEP

FP

Fractional Part Function: Returns the fractional part of the argument.

{ }

Level 1	→	Level 1
x	→	*y*
x_unit	→	*y_unit*
'*symb*'	→	'FP(*symb*)'

Keyboard Access: (MTH) REAL (NXT) FP

Affected by Flags: Numerical Results (−3)

Remarks: The result has the same sign as the argument.

Examples: `-32.3 FP` returns `-.3`; `32.3_m FP` returns `.3_m`.

Related Commands: IP

FREE

Free RAM Card Command: Frees (makes *independent*) the previously merged RAM in port 1. FREE is provided for compatibility with the HP 48SX, which could merge RAM in port 2 as well. See FREE1.

Level 2	Level 1	→	Level 1
{ }	n_{port}	→	
{$name_{backup}$ ⋯ $n_{library}$ }	n_{port}	→	
$name_{backup}$	n_{port}	→	
$n_{library}$	n_{port}	→	

FREE

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Any prior contents of the port are moved into user memory. If level 2 specifies backup or library objects, those objects are moved from port 0 to the newly freed RAM port.

Related Commands: FREE1, MERGE1

FREE1

Free RAM Card Command: Frees (makes *independent*) the previously merged RAM in port 1. Any prior contents of the port are moved into user memory. If level 1 specifies backup or library objects, those objects are moved from port 0 to the newly freed RAM port.

Level 1	→	Level 1
$\{ name_{backup} \cdots n_{library} \}$	→	
$name_{backup}$	→	
$n_{library}$	→	

Keyboard Access: ↩ LIBRARY FREE1

Affected by Flags: None

Remarks: The list in level 1 can be empty (in which case no objects are moved to the newly independent RAM) or it can contain any number of backup names and library numbers. Level 1 cannot be completely empty, however.

Related Commands: FREE, MERGE, MERGE1

FREEZE

Freeze Display Command: Freezes the part of the display specified by $n_{\text{display area}}$, so that it is not updated until a key is pressed.

{}

Level 1	→	Level 1
$n_{\text{displayarea}}$	→	

Keyboard Access: (PRG) (NXT) OUT FREEZ

Affected by Flags: None

Remarks: Normally, the stack display is updated as soon as the calculator is ready for data input. For example, when HALT stops a running program, or when a program ends, any displayed messages are cleared. The FREEZE command "freezes" a part or all of the display so that it is not updated until a key is pressed. This allows, for example, a prompting message to persist after a program halts to await data input.

$n_{\text{display area}}$ is the sum of the value codes for the areas to be frozen:

Display Area	Value Code
Status area	1
Stack/Command-line area	2
Menu area	4

So, for example, 2 FREEZE freezes the stack/command-line area, 3 FREEZE freezes the status area and the stack/command-line area, and 7 FREEZE freezes all three areas.

Values of $n_{\text{display area}} \geq 7$ or ≤ 0 freeze the entire display (are equivalent to value 7). To freeze the graphics display, you must freeze the status and stack/command-line areas (by entering 3), or the entire display (by entering 7).

FREEZE

Examples:

This program:

```
« "Ready for data" 1 DISP 1 FREEZE HALT »
```

displays the contents of the string in the top line of the display, then freezes the status area so that the string contents persist in the display after HALT is executed:

This program:

```
« { # 0d # 0d } PVIEW 7 FREEZE »
```

selects the graphics display and then freezes the entire display so that the graphics display persists after the program ends. (If FREEZE was not executed, the stack display would be selected after the program ends.)

To use FREEZE with PVIEW (or any graphics display), you must enter 3 or 7.

Related Commands: CLLCD, DISP, HALT

FS?

Flag Set? Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is set, and returns a corresponding test result: 1 (true) if the flag is set or 0 (false) if the flag is clear.

{ }

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	0/1

Keyboard Access:

PRG TEST NXT NXT FS?

◀ MODES FLAG FS?

Affected by Flags: None

Related Commands: CF, FC?, FC?C, FS?C, SF

FS?C

Flag Set? Clear Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is set, and returns a corresponding test result: 1 (true) if the flag is set or 0 (false) if the flag is clear. After testing, clears the flag.

{ }

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	0/1

Keyboard Access:

PRG TEST NXT NXT FS?C

MODES FLAG FS?C

Affected by Flags: None

Example: If flag −44 is set, −44 FS?C returns 1 to level 1 and clears flag −44.

Related Commands: CF, FC?, FC?C, FS?, SF

FUNCTION

Function Plot Type Command: Sets the plot type to FUNCTION.

Keyboard Access: MODES PLOT PTYPE FUNC

Affected by Flags: Simultaneous Plotting (−28), Curve Filling (−31)

Remarks: When the plot type is FUNCTION, the DRAW command plots the current equation as a real-valued function of one real variable. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has the form:

$$\{\ \{x_{min}\ y_{min}\}\ \{x_{max}\ y_{max}\}\ indep\ res\ axes\ ptype\ depend\ \}$$

FUNCTION

For plot type FUNCTION, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (-6.5,-3.1).
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value of *indep* is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command FUNCTION places the name FUNCTION in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The current equation is plotted as a function of the variable specified in *indep*. The minimum and maximum values of the independent variable (the plotting range) can be specified in *indep*; otherwise, the values in (x_{min}, y_{min}) and (x_{max}, y_{max}) (the display range) are used. Lines are drawn between plotted points unless flag −31 is set.

If *EQ* contains an expression or program, the expression or program is evaluated in Numerical Results mode for each value of the independent variable to give the values of the dependent variable. If *EQ* contains an equation, the plotting action depends on the form of the equation, as shown in the following table.

Form of Current Equation	Plotting Action
'*expr*=*expr*'	Each expression is plotted separately. The intersection of the two graphs shows where the expressions are equal.
'*name*=*expr*'	Only the expression is plotted.
'*indep*=*constant*'	A vertical line is plotted.

If flag −28 is set, all equations are plotted simultaneously.

If the independent variable in the current equation represents a unit object, you must specify the units by storing a unit object in the corresponding variable in the current directory. For example, if the current equation is '`X+3_m`', and you want X to represent some number of inches, you would store `1_in` (the number part of the unit object is ignored) in X. For each plotted point, the numerical value of the independent variable is combined with the specified unit (inches in this example) before the current equation is evaluated. If the result is a unit object, only the number part is plotted.

Related Commands: BAR, CONIC, DIFFEQ, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

GET

Get Element Command: Returns from the level 2 array or list (or named array or list) the real or complex number $z_{\texttt{get}}$ or object $obj_{\texttt{get}}$ whose position is specified in level 1.

GET

Level 2	Level 1	→	Level 1
[[*matrix*]]	$n_{position}$	→	z_{get}
[[*matrix*]]	{ n_{row} m_{col} }	→	z_{get}
'$name_{matrix}$'	$n_{position}$	→	z_{get}
'$name_{matrix}$'	{ n_{row} m_{col} }	→	z_{get}
[*vector*]	$n_{position}$	→	z_{get}
[*vector*]	{ $n_{position}$ }	→	z_{get}
'$name_{vector}$'	$n_{position}$	→	z_{get}
'$name_{vector}$'	{ $n_{position}$ }	→	z_{get}
{ *list* }	$n_{position}$	→	obj_{get}
{ *list* }	{ $n_{position}$ }	→	obj_{get}
'$name_{list}$'	$n_{position}$	→	obj_{get}
'$name_{list}$'	{ $n_{position}$ }	→	obj_{get}

Keyboard Access: PRG LIST ELEM GET

Affected by Flags: None

Remarks: For matrices, $n_{position}$ is incremented in *row* order.

Examples: [[2 8 7][3 2 9][2 1 8]] { 2 8 } GET returns 9.

[[2 8 7][3 2 9][2 1 8]] 8 GET returns 1.

{ A B C D E } { 1 } GET returns 'A'.

Related Commands: GETI, PUT, PUTI

GETI

Get and Increment Index Command: Returns from the level 2 array or list (or named array or list) the real or complex number z_{get} or object obj_{get} whose position is specified in level 1, along with the level 2 argument and the next position in that argument.

Level 2	Level 1	→	Level 3	Level 2	Level 1
[[*matrix*]]	n_{pos1}	→	[[*matrix*]]	n_{pos2}	z_{get}
[[*matrix*]]	$\{ n_r \; m_c \}_1$	→	[[*matrix*]]	$\{ n_r \; m_c \}_2$	z_{get}
'$name_{mtrx}$'	n_{pos1}	→	'$name_{mtrx}$'	n_{pos2}	z_{get}
'$name_{mtrx}$'	$\{ n_r \; m_c \}_1$	→	'$name_{mtrx}$'	$\{ n_r \; m_c \}_2$	z_{get}
[*vect*]	n_{pos1}	→	[*vect*]	n_{pos2}	z_{get}
[*vect*]	$\{ n_{pos1} \}$	→	[*vect*]	$\{ n_{pos2} \}$	z_{get}
'$name_{vect}$'	n_{pos1}	→	'$name_{vect}$'	n_{pos2}	z_{get}
'$name_{vect}$'	$\{ n_{pos1} \}$	→	'$name_{vect}$'	$\{ n_{pos2} \}$	z_{get}
{ *list* }	n_{pos1}	→	{ *list* }	n_{pos2}	obj_{get}
{ *list* }	$\{ n_{pos1} \}$	→	{ *list* }	$\{ n_{pos2} \}$	obj_{get}
'$name_{list}$'	n_{pos1}	→	'$name_{list}$'	n_{pos2}	obj_{get}
'$name_{list}$'	$\{ n_{pos1} \}$	→	'$name_{list}$'	$\{ n_{pos2} \}$	obj_{get}

Keyboard Access: PRG LIST ELEM GETI

Affected by Flags: Index Wrap Indicator (−64)

The Index Wrap Indicator flag is cleared on each execution of GETI *until* the position index wraps to the first position in the argument, at which point the flag is set. The next execution of GETI clears the flag again.

Remarks: For matrices, the position is incremented in *row* order.

Related Commands: GET, PUT, PUTI

GOR

Graphics OR Command: Superimposes $grob_1$ onto $grob_{target}$ or $PICT$, with the upper left corner pixel of $grob_1$ positioned at the specified coordinate in $grob_{target}$ or $PICT$.

Level 3	Level 2	Level 1	→	Level 1
$grob_{target}$	{ #n #m }	$grob_1$	→	$grob_{result}$
$grob_{target}$	(x,y)	$grob_1$	→	$grob_{result}$
PICT	{ #n #m }	$grob_1$	→	
PICT	(x,y)	$grob_1$	→	

Keyboard Access: PRG GROB GOR

Affected by Flags: None

Remarks: GOR uses a logical OR to determine the state (on or off) of each pixel in the overlapping portion of the argument graphics object.

If the level 3 argument is any graphics object other than *PICT*, then $grob_{result}$ is returned to the stack. If the level 3 argument is *PICT*, no result is returned to the stack.

Any portion of $grob_1$ that extends past $grob_{target}$ or *PICT* is truncated.

Related Commands: GXOR, REPL, SUB

GRAD

Grads Mode Command: Sets Grads angle mode.

Keyboard Access: ◀ MODES ANGL GRAD

Affected by Flags: None

Remarks: GRAD clears flag −17 and sets flag −18, and displays the GRAD annunciator.

In Grads angle mode, real-number arguments that represent angles are interpreted as grads, and real-number results that represent angles are expressed in grads.

Related Commands: DEG, RAD

GRAPH

Picture Environment Command: Selects the Picture environment (selects the graphics display and activates the graphics cursor and Picture menu).

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: GRAPH is provided for compatibility with the HP 48S series, and is the same as the PICTURE command. See PICTURE.

Related Commands: PICTURE, PVIEW, TEXT

GRIDMAP

GRIDMAP Plot Type Command: Sets plot type to GRIDMAP.

Keyboard Access: ◀ PLOT NXT 3D PTYPE GRID

Affected by Flags: None

Remarks: When plot type is set GRIDMAP, the DRAW command plots a mapping grid representation of a 2-vector-valued function of two variables. GRIDMAP requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

{ x_{left} x_{right} y_{near} y_{far} z_{low} z_{high} x_{min} x_{max} y_{min} y_{max} x_{eye} y_{eye} z_{eye} x_{step} y_{step} }.

For plot type GRIDMAP, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are real numbers the specify the input region's width. The default value is (−1, 1).
- y_{min} and y_{max} are real numbers that specify the input region's depth. The default value is (−1, 1).
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which you view the graph.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted. These can be used instead of (or in combination with) RES.

The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

For plot type GRIDMAP, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is not used.

- (x_{max}, y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is X.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command GRIDMAP places the command name GRIDMAP in *PPAR*.
- *depend* is a name specifying the dependent variable. The default value is Y.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

→GROB

Stack to Graphics Object Command: Creates a graphics object representing the level 2 object, where the argument $n_{char\ size}$ specifies the character size of the representation.

{ }

Level 2	Level 1	→	Level 1
obj	$n_{charsize}$	→	*grob*

Keyboard Access: PRG GROB →GRO

Affected by Flags: None

Remarks: $n_{char\ size}$ can be 0, 1 (small), 2 (medium), or 3 (large). $n_{char\ size} = 0$ is the same as $n_{char\ size} = 3$, except for unit objects

and algebraic objects, where 0 specifies the EquationWriter application picture.

Example: This program:

```
« 'Y=3*X^2' 0 →GROB PICT STO { } PVIEW »
```

returns a graphics object to the stack representing the EquationWriter application picture of 'Y=3*X^2', then stores the graphics object in *PICT* and shows it in the graphics display with scrolling activated.

Related Commands: →LCD, LCD→

GXOR

Graphics Exclusive OR Command: Superimposes $grob_1$ onto $grob_{\text{target}}$ or *PICT*, with the upper left corner pixel of $grob_1$ positioned at the specified coordinate in $grob_{\text{target}}$ or *PICT*.

Level 3	Level 2	Level 1	→	Level 1
$grob_{\text{target}}$	{ #n #m }	$grob_1$	→	$grob_{\text{result}}$
$grob_{\text{target}}$	(x,y)	$grob_1$	→	$grob_{\text{result}}$
PICT	{ #n #m }	$grob_1$	→	
PICT	(x,y)	$grob_1$	→	

Keyboard Access: PRG GROB GXOR

Affected by Flags: None

Remarks: GXOR is used for creating cursors, for example, to make the cursor image appear dark on a light background and light on a dark background. Executing GXOR again with the same image restores the original picture.

GXOR uses a logical exclusive OR to determine the state of the pixels (on or off) in the overlapping portion of the argument graphics objects.

Any portion of $grob_1$ that extends past $grob_{target}$ or *PICT* is truncated.

If the level 3 argument (the target graphics object) is any graphics object other than *PICT*, then $grob_{result}$ is returned to the stack. If the level 3 argument is *PICT*, no result is returned to the stack.

Example: This program:

```
« ERASE PICT NEG PICT { # 0d #0d }
GROB 5 × 5 11A040A011 GXOR LASTARG GXOR »
```

turns on (makes dark) every pixel in *PICT*, then superimposes a 5 × 5 graphics object on *PICT* at pixel coordinates { # 0d # 0d }. Each on-pixel in the 5 by 5 graphics object turns off (makes light) the corresponding pixel in *PICT*. Then, the original picture is restored by executing GXOR again with the same arguments.

Related Commands: GOR, REPL, SUB

---

*H

Multiply Height Command: Multiplies the vertical plot scale by x_{factor}.

{ }

Level 1	→	Level 1
x_{factor}	→	

Keyboard Access: ↰ PLOT FFHF NXT *H

Affected by Flags: None

Remarks: Executing *H changes the y-axis display range—the y_{min} and y_{max} components of the first two complex numbers in the reserved variable *PPAR*. The plot origin (the user-unit coordinate of the center pixel) is not changed.

Related Commands: AUTO, *W, YRNG

HALT

Halt Program Command: Halts program execution.

Keyboard Access: PRG NXT RUN HALT

Affected by Flags: None

Remarks: Program execution is halted at the location of the HALT command in the program. The HALT annunciator is turned on. Program execution is resumed by executing CONT (usually by pressing ← CONT). Executing KILL (usually by pressing PRG NXT RUN KILL) cancels all halted programs.

Related Commands: CONT, KILL

HEAD

First Listed Element Command: Returns the first element of a list or string.

Level 1	→	Level 1
{ obj_1 ... obj_n }	→	obj_1
"*string*"	→	"$element_1$"

Keyboard Access:

PRG LIST ELEM NXT HEAD

← CHARS NXT HEAD

Affected by Flags: None

Example: "Dead" HEAD returns "D".

The following program takes a list of coordinates { A B C } that define a right triangle, and finds the length of the hypotenuse AC:

```
« DUP HEAD SWAP REVLIST HEAD - ABS »
```

For example, entering { (0,0) (0,3) (3,4) } returns 5.

Related Commands: TAIL

HEX

Hexadecimal Mode Command: Selects hexadecimal base for binary integer operations. (The default base is decimal.)

Keyboard Access: (MTH) BASE HEX

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: Binary integers require the prefix #. Binary integers entered and returned in hexadecimal base automatically show the suffix h. If the current base is not hexadecimal, then you can enter a hexadecimal number by ending it with h. It will be displayed in the current base when it is entered.

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Related Commands: BIN, DEC, OCT, RCWS, STWS

HISTOGRAM

Histogram Plot Type Command: Sets the plot type to HISTOGRAM.

Keyboard Access: ← (PLOT) (NXT) STAT PTYPE HISTO

Affected by Flags: None

Remarks: When the plot type is HISTOGRAM, the DRAW command creates a histogram using data from one column of the current statistics matrix (reserved variable ΣDAT). The column is specified by the first parameter in the reserved variable ΣPAR (using the XCOL command). The plotting parameters are specified in the reserved variable $PPAR$, which has the form:

$$\{\ (x_{min}, y_{min})\ (x_{max}, y_{max})\ indep\ res\ axes\ ptype\ depend\ \}$$

HISTOGRAM

For plot type HISTOGRAM, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (-6.5,-3.1).
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is either a name specifying a label for the horizontal axis, or a list containing such a name and two numbers that specify the minimum and maximum values of the data to be plotted. The default value of *indep* is *X*.
- *res* is a real number specifying the bin size, in user-unit coordinates, or a binary integer specifying the bin size in pixels. The default value is 0, which specifies the bin size to be 1/13 of the difference between the specified minimum and maximum values of the data.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command HISTOGRAM places the command name HISTOGRAM in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The frequency of the data is plotted as bars, where each bar represents a collection of data points. The base of each bar spans the values of the data points, and the height indicates the number of data points. The width of each bar is specified by *res*. The overall maximum and minimum values for the data can be specified by *indep*; otherwise, the values in (x_{min}, y_{min}) and (x_{max}, y_{max}) are used.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

HISTPLOT

Draw Histogram Plot Command: Plots a frequency histogram of the specified column in the current statistics matrix (reserved variable ΣDAT).

Keyboard Access: ↰ STAT PLOT HISTP

Affected by Flags: None

Remarks: The data column to be plotted is specified by XCOL and is stored as the first parameter in the reserved variable ΣPAR. If no data column is specified, column 1 is selected by default. The y-axis is autoscaled and the plot type is set to HISTOGRAM.

HISTPLOT plots *relative* frequencies, using 13 bins as the default number of partitions. The RES command lets you specify a different number of bins by specifying the bin width. To plot a frequency histogram with *numerical* frequencies, store the frequencies in ΣDAT and execute BINS and then BARPLOT.

When HISTPLOT is executed from a program, the graphics display, which shows the resultant plot, does not persist unless PICTURE, PVIEW (with an empty list argument), or FREEZE is subsequently executed.

Related Commands: BARPLOT, BINS, FREEZE, PICTURE, PVIEW, RES, SCATRPLOT, XCOL

HMS+

Hours-Minutes-Seconds Plus Command: Returns the sum of two real numbers, where the arguments and the result are interpreted in hours-minutes-seconds format.

{ }

Level 2	Level 1	→	Level 1
HMS_1	HMS_2	→	$HMS_1 + HMS_2$

HMS+

Keyboard Access: ↰ TIME NXT HMS+

Affected by Flags: None

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: HMS→, →HMS, HMS−

HMS−

Hours-Minutes-Seconds Minus Command: Returns the difference of two real numbers, where the arguments and the result are interpreted in hours-minutes-seconds format.

{ }

Level 2	Level 1	→	Level 1
HMS_1	HMS_2	→	$HMS_1 - HMS_2$

Keyboard Access: ↰ TIME NXT HMS−

Affected by Flags: None

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: HMS→, →HMS, HMS+

HMS→

Hours-Minutes-Seconds to Decimal Command: Converts a real number in hours-minutes-seconds format to its decimal form (hours or degrees with a decimal fraction).

{ }

Level 1	→	Level 1
HMS	→	*x*

Keyboard Access: ↰ TIME NXT HMS+

Affected by Flags: None

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: →HMS, HMS+, HMS−

→HMS

Decimal to Hours-Minutes-Seconds Command: Converts a real number representing hours or degrees with a decimal fraction to hours-minutes-seconds format.

{ }

Level 1	→	Level 1
x	→	*HMS*

Keyboard Access: ↰ TIME NXT HMS→

Affected by Flags: None

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: HMS→, HMS+, HMS−

HOME

HOME Directory Command: Makes the *HOME* directory the current directory.

Keyboard Access: ↱ HOME

Affected by Flags: None

Related Commands: CRDIR, PATH, PGDIR, UPDIR

i

i Function: Returns the symbolic constant i or its numerical representation, (0, 1).

Level 1	→	Level 1
	→	'i'
	→	(0,1)

Keyboard Access:

α ↰ I

MTH NXT CONS I

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

Evaluating i returns its numerical representation if flag −2 or −3 is set; otherwise, its symbolic representation is returned.

Related Commands: e, MAXR, MINR, π

IDN

Identity Matrix Command: Returns an identity matrix; that is, a square matrix with its diagonal elements equal to 1 and its off-diagonal elements equal to 0.

{ }

Level 1	→	Level 1
n	→	[[$R\text{-}matrix_{identity}$]]
[[*matrix*]]	→	[[$matrix_{identity}$]]
'*name*'	→	

IDN

Keyboard Access: (MTH) MATR MAKE IDN

Affected by Flags: None

Remarks: The result is either a new square matrix, or it's an existing square matrix with its elements replaced by the elements of the identity matrix, according to the argument in level 1.

- **Creating a new matrix:** If the argument is a real number n, a new real identity matrix is returned to level 1, with its number of rows and number of columns equal to n.
- **Replacing the elements of an existing matrix:** If the argument is a square matrix, an identity matrix of the same dimensions is returned. If the original matrix is complex, the resulting identity matrix will also be complex, with diagonal values (1,0).

  If the argument is a name, the name must identify a variable containing a square matrix. In this case, the elements of the matrix are replaced by those of the identity matrix (complex if the original matrix is complex).

Related Commands: CON

IF

IF Conditional Structure Command: Starts IF ... THEN ... END and IF ... THEN ... ELSE ... END conditional structures.

	Level 1	→	Level 1
IF		→	
THEN	*T/F*	→	
END		→	
		→	
IF		→	
THEN	*T/F*	→	
ELSE		→	
END		→	

Keyboard Access: (PRG) BRCH IF IF

Affected by Flags: None

Remarks: *Conditional structures*, used in combination with program tests, enable a program to make decisions.

- IF ... THEN ... END executes a sequence of commands only if a test returns a nonzero (true) result. The syntax is:

  IF *test-clause* THEN *true-clause* END

  IF begins the test clause, which must return a test result to the stack. THEN removes the test result from the stack. If the value is nonzero, the true clause is executed. Otherwise, program execution resumes following END.

- IF ... THEN ... ELSE ... END executes one sequence of commands if a test returns a true (nonzero) result, or another sequence of commands if that test returns a false (zero) result. The syntax is:

  IF *test-clause* THEN *true-clause* ELSE *false-clause* END

  IF begins the test clause, which must return a test result to the stack. THEN removes the test result from the stack. If the value is nonzero, the true clause is executed. Otherwise, the false clause is executed. After the appropriate clause is executed, execution resumes following END.

The test clause can be a command sequence (for example, A B ≤) or an algebraic (for example, 'A≤B'). If the test clause is an algebraic, it is *automatically evaluated* to a number (→NUM or EVAL isn't necessary).

Related Commands: CASE, ELSE, END, IFERR, THEN

IFERR

If Error Conditional Structure Command Starts IFERR ... THEN ... END and IFERR ... THEN ... ELSE ... END error trapping structures.

Keyboard Access: PRG NXT ERROR IFERR

Affected by Flags: Last Arguments (−55)

Remarks: *Error trapping* structures enable program execution to continue after a "trapped" error occurs.

- IFERR ... THEN ... END executes a sequence of commands if an error occurs. The syntax of IFERR ... THEN ... END is:

  IFERR *trap-clause* THEN *error-clause* END

  If an error occurs during execution of the trap clause:

  1. The error is ignored.
  2. The remainder of the trap clause is discarded.
  3. The key buffer is cleared.
  4. If any or all of the display is "frozen" (by FREEZE), that state is canceled.
  5. If Last Arguments is enabled, the arguments to the command that caused the error are returned to the stack.
  6. Program execution jumps to the error clause.

  The commands in the error clause are executed only if an error is generated during execution of the trap clause.

- IFERR ... THEN ... ELSE ... END executes one sequence of commands if an error occurs or another sequence of commands if an error does not occur. The syntax of IFERR ... THEN ... ELSE ... END is:

  IFERR *trap-clause* THEN *error-clause* ELSE *normal-clause* END

  If an error occurs during execution of the trap clause, the same six events listed above occur.

  If no error occurs, execution jumps to the normal clause at the completion of the trap clause.

Example: The following program uses IFERR much like the built-in linear system of equations solver. The program takes a result vector and a matrix of coefficients and returns a least-squares solution to the equations.

```
« → a b
  « IFERR a b / THEN LSQ END
  »
»
```

Related Commands: CASE, ELSE, END, IF, THEN

IFFT

Inverse Discrete Fourier Transform Command: Computes the one- or two-dimensional inverse discrete Fourier transform of an array.

{ }

Level 1	→	Level 1
[*array*]$_1$	→	[*array*]$_2$

Keyboard Access: MTH NXT FFT IFFT

Affected by Flags: None

Remarks: If the argument is an N-vector or an $N \times 1$ or $1 \times N$ matrix, IFFT computes the one-dimensional inverse transform. If the argument is an $M \times N$ matrix, IFFT computes the two-dimensional inverse transform. M and N must be integral powers of 2.

The one-dimensional inverse discrete Fourier transform of an N-vector Y is the N-vector X where:

$$X_n = \frac{1}{N} \sum_{k=0}^{N-1} Y_k e^{\frac{2\pi ikn}{N}} , \qquad i = \sqrt{-1}$$

for $n = 0,\ 1,\ \ldots,\ N-1$.

IFFT

The two-dimensional inverse discrete Fourier transform of an $M \times N$ matrix Y is the $M \times N$ matrix X where:

$$X_{mn} = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} Y_{kl} e^{\frac{2\pi i k m}{M}} e^{\frac{2\pi i l n}{N}}, \qquad i = \sqrt{-1}$$

for $m = 0,\ 1,\ ...,\ M-1$ and $n = 0,\ 1,\ ...,\ N-1$.

The discrete Fourier transform and its inverse are defined for any positive sequence length. However, the calculation can be performed very rapidly when the sequence length is a power of two, and the resulting algorithms are called the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT).

The IFFT command uses truncated 15-digit arithmetic and intermediate storage, then rounds the result to 12-digit precision.

Related Commands: FFT

IFT

IF-THEN Command: Executes *obj* if *T/F* is nonzero. Discards *obj* if *T/F* is zero.

Level 2	Level 1	→	Level 1
T/F	*obj*	→	*It depends!*

Keyboard Access: PRG BRCH NXT IFT

Affected by Flags: None

Remarks: IFT lets you execute in stack syntax the decision-making process of the IF ... THEN ... END conditional structure. The "true clause" is *obj* in level 1.

Example: `« X 0 > "Positive" IFT »` puts `"Positive"` in level 1 if X contains a positive real number.

Related Commands: IFTE

IFTE

IF-THEN-ELSE Function: Executes the *obj* on level 2 if *T/F* is nonzero. Executes the *obj* on level 1 if *T/F* is zero.

Level 3	Level 2	Level 1	→	Level 1
T/F	obj_{true}	obj_{false}	→	*It depends!*

Keyboard Access: (PRG) BRCH (NXT) IFTE

Affected by Flags: None

Remarks: IFTE lets you execute in stack syntax the decision-making process of the IF ... THEN ... ELSE ... END conditional structure. The "true clause" is obj_{true} in level 2. The "false clause" is obj_{false} in level 1.

IFTE is also allowed in algebraic expressions, with the following syntax:

'IFTE(*test*,*true-clause*,*false-clause*)'

When an algebraic containing IFTE is evaluated, its first argument *test* is evaluated to a test result. If it returns a nonzero real number, *true-clause* is evaluated. If it returns zero, *false-clause* is evaluated.

Examples: The command sequence `X 0 ≥ "Positive" "Negative" IFTE` leaves `"Positive"` on the stack if X contains a non-negative real number, or `"Negative"` if X contains a negative real number.

The algebraic `'IFTE(X≠0,SIN(X)/X,1)'` returns the value of $\sin(x)/x$, even for $x = 0$, which would normally cause an `Infinite Result` error.

Related Commands: IFT

IM

Imaginary Part Function: Returns the imaginary part of its (complex) argument.

{ }

Level 1	→	Level 1
x	→	0
(*x*, *y*)	→	*y*
[*R-array*]	→	[*R-array*]
[*C-array*]	→	[*R-array*]
'*symb*'	→	'IM(*symb*)'

Keyboard Access: MTH NXT CMPL IM

Affected by Flags: Numerical Results (−3)

Remarks: If the argument is an array, IM returns a real array, the elements of which are equal to the imaginary parts of the corresponding elements of the argument array. If the argument array is real, all of the elements of the result array are zero.

Related Commands: C→R, RE, R→C

INCR

Increment Command: Takes a variable on level 1, adds 1, stores the new value back into the original variable, and returns the new value to level 1.

{ }

Level 1	→	Level 1
'*name*'	→	$x_{increment}$

Keyboard Access: ◀ MEMORY ARITH INCR

Affected by Flags: None

Remarks: The value in *name* must be a real number.

Example: If 35.7 is stored in *A*, 'A' INCR returns 36.7.

Related Commands: DECR

INDEP

Independent Variable Command: Specifies the independent variable and its plotting range.

Level 2	Level 1	→	Level 1
	'*global*'	→	
	{ *global* }	→	
	{ *global* x_{start} x_{end} }	→	
	{ x_{start} x_{end} }	→	
x_{start}	x_{end}	→	

Keyboard Access: ◀ PLOT FPAR INDEP

Affected by Flags: None

Remarks: The specification for the independent variable name and its plotting range is stored as the third parameter in the reserved variable *PPAR*. If the argument to INDEP is a:

- Global variable name, that name replaces the independent variable entry in *PPAR*.
- List containing a global name, that name replaces the independent variable name but leaves unchanged any existing plotting range.
- List containing a global name and two real numbers, that list replaces the independent variable entry.

INDEP

- List containing two real numbers, or two real numbers from levels 1 and 2, those two numbers specify a new plotting range, leaving the independent variable name unchanged. (LASTARG returns a list, even if the two numbers were entered separately.)

The default entry is *X*.

Related Commands: DEPND

INFORM

User-Defined Dialog Box Command: Creates a user-defined input form (dialog box).

Lvl 5	Lvl 4	Lvl 3	Lvl 2	Lvl 1 →	Lvl 2	Lvl 1
"*title*"	{ s_1 s_2 ... s_n }	*format*	{ *resets* }	{ *init* } →	{ *vals* }	*1*
"*title*"	{ s_1 s_2 ... s_n }	*format*	{ *resets* }	{ *init* } →		0

Keyboard Access: PRG NXT IN INFOR

Affected by Flags: None

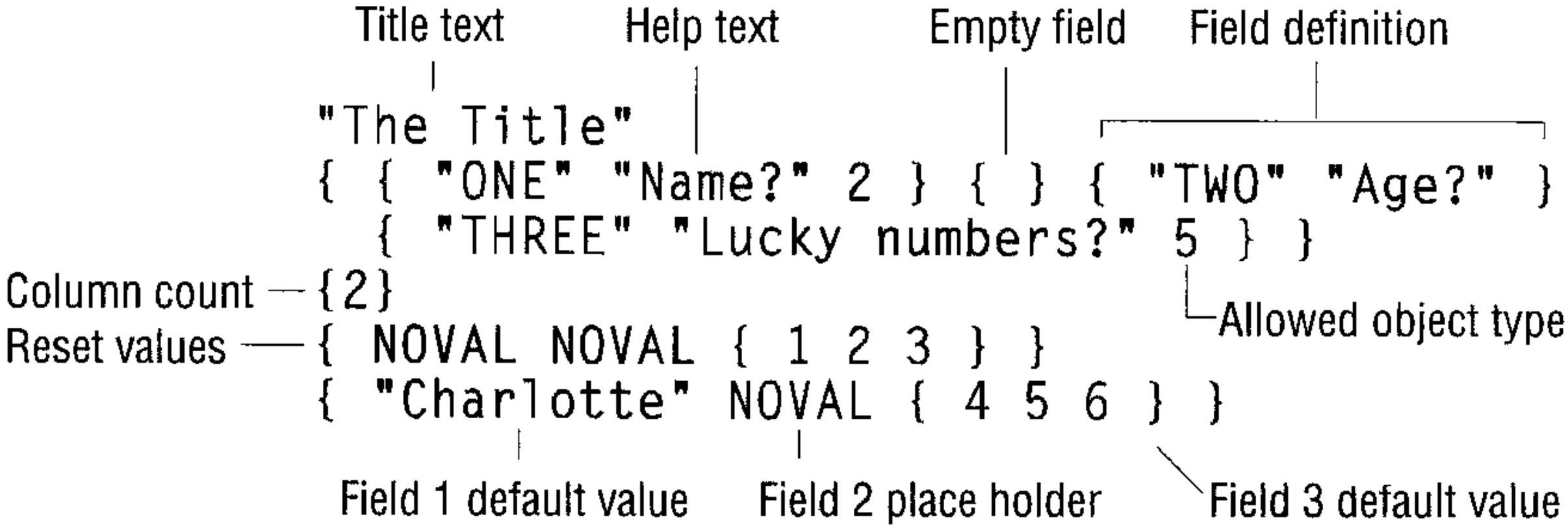
Remarks: INFORM creates a standard dialog box based upon the following specifications:

Variable	Function
"title"	Title. This appears at the top of the dialog box.
$\{s_1\ s_2\ \ldots\ s_n\}$	Field definitions. A field definition (s_x) can have two formats: *"label"*, a field label, or { *"label" "helpInfo" type$_0$ type$_1$ ... type$_n$* }, a field label with optional help text that appears near the bottom of the screen, and an optional list of valid object types for that field. If object types aren't specified, all object types are valid. For infomation about object types, see the TYPE command. When creating a multi-column dialog box, you can span columns by using an empty list as a field definition. A field that appears to the left of an empty field automatically expands to fill the empty space.
format	Field format information. This is the number *col* or a list of the form { *col tabs* }: *col* is the number of columns the dialog box has, and *tabs* optionally specifies the number of tab stops between the labels and the highlighted fields. This list can be empty. *col* defaults to 1 and *tabs* defaults to 3.
{ *resets* }	Default values displayed when RESET is selected. Specify reset values in the list in the same order as the fields were specified. To specify no value, use the NOVAL command as a place holder. This list can be empty.
{ *init* }	Initial values displayed when the dialog box appears. Specify initial values in the list in the same order as the fields were specified. To specify no value, use the NOVAL command as a place holder. This list can be empty.

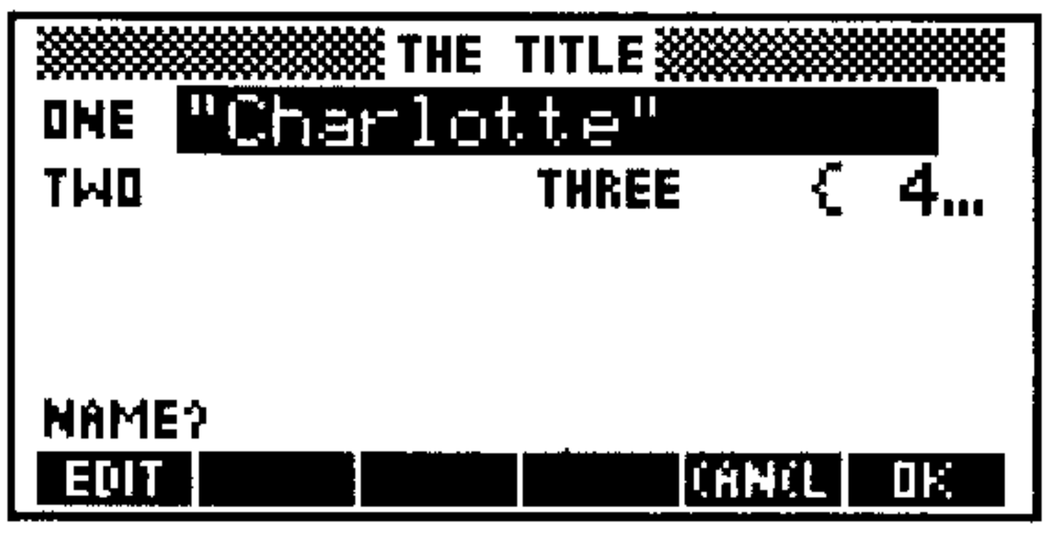
If you exit the dialog box by selecting OK or ENTER, INFORM returns the field values { *vals* } on level 2, and puts a 1 on level 1. (If a field is empty, NOVAL is returned as a place holder.) If you exit the dialog box by selecting CANCEL or CANCEL, INFORM returns 0.

INFORM

Example: If the following five lines are on the stack:



Pressing INFORM would produce:



Related Commands: CHOOSE, INPUT, NOVAL, TYPE

INPUT

Input Command: Prompts for data input to the command line and prevents the user access to stack operations.

Level 2	Level 1	→	Level 1
"stack prompt"	*"command-line prompt"*	→	*"result"*
"stack prompt"	$\{\ list_{command-line}\ \}$	→	*"result"*

Keyboard Access: PRG NXT IN INPUT

Affected by Flags: None

Remarks: When INPUT is executed, the stack area is blanked and program execution is suspended for data input to the command line. The contents of "*stack prompt*" are displayed at the top of the stack area. Depending on the level 1 argument, the command line may also contain the contents of a string, or it may be empty. Pressing ENTER resumes program execution and returns the contents of the command line in string form to level 1.

In its general form, the level 1 argument for INPUT is a list that specifies the content and interpretation of the command line. The list can contain *one or more* of the following parameters, *in any order*:

- "*command-line prompt*", whose contents are placed in the command line for prompting when the program pauses.
- Either a *real number*, or a *list containing two real numbers*, that specifies the initial cursor position in the command line:
  - A real number *n* at the *n*th character from the left end of the first row (line) of the command line. A *positive n* specifies the insert cursor; a *negative n* specifies the replace cursor. ⊡ specifies the end of the command-line string.
  - A list that specifies the initial row and column position of the cursor: the first number in the list specifies a row in the command line (1 specifies the first row of the command line); the second number counts by characters from the left end of the specified line. ⊡ specifies the end of the command-line string in the specified row. A positive row number specifies the insert cursor; a negative row number specifies the replace cursor.
- One or more of the parameters ALG, α, or V, entered as unquoted names:
  - ALG activates Algebraic/Program-entry mode.
  - α (α → A) specifies alpha lock.
  - V verifies if the characters in the result string "result", without the " delimiters, compose a valid object or objects. If the result-string characters do not compose a valid object or objects, INPUT displays the Invalid Syntax warning and prompts again for data.

INPUT

You can choose to specify as few as one of the level-1 list parameters. The default states for these parameters are:

- Blank command line.
- Insert cursor placed at the end of the command-line prompt string.
- Program-entry mode.
- Result string not checked for invalid syntax.

If you specify *only* a command-line prompt string for the level 1 argument, you don't need to put it in a list.

Related Commands: PROMPT, STR→

INV

Inverse (1/x) Analytic Function: Returns the reciprocal or the matrix inverse.

{ }

Level 1	→	Level 1
z	→	*1/z*
[[*matrix*]]	→	[[*matrix*]] $^{-1}$
'*symb*'	→	'INV(*symb*)'
x_unit	→	*1/x_1/unit*

Keyboard Access: (1/x)

Affected by Flags: Numerical Results (−3)

Remarks: For a *complex* argument (x, y), the inverse is the complex number $\left(\frac{x}{x^2+y^2}, \frac{-y}{x^2+y^2}\right)$.

Matrix arguments must be square (real or complex). The computed inverse matrix A^{-1} satisfies $A \times A^{-1} = I_n$, where I_n is the $n \times n$ identity matrix.

Related Commands: SINV, /

IP

Integer Part Function: Returns the integer part of its argument.

{ }

Level 1	→	Level 1
x	→	*n*
x_unit	→	*n_unit*
'*symb*'	→	'IP(*symb*)'

Keyboard Access: (MTH) REAL (NXT) IP

Affected by Flags: Numerical Results (−3)

Remarks: The result has the same sign as the argument.

Example: 32.3_m IP returns 32_m.

Related Commands: FP

IR

Infrared/Serial Transmission Command: Directs I/O and printer output to the infrared or serial port ("wire").

Keyboard Access: ↰ (I/O) I/O FF IR

Affected by Flags: I/O Device (−33), Printing Device (−34)

Remarks: Toggles between IR and wire.

For more information, refer also to the reserved variable *IOPAR* (*I/O parameters*) in appendix D, "Reserved Variables."

Related Commands: BAUD, CKSM, PARITY, TRANSIO

ISOL

Isolate Variable Command: Returns an algebraic '$symb_2$' that rearranges '$symb_1$' to "isolate" the first occurrence of variable *global*.

{}

Level 2	Level 1	→	Level 1
'$symb_1$'	'*global*'	→	'$symb_2$'

Keyboard Access: ↩ SYMBOLIC ISOL

Affected by Flags: Principal Solution (−1), Numerical Results (−3)

When flag −3 is set, symbolic results are evaluated to real numbers. This means that the = sign is evaluated. If *global* or any other variable in the result equation is formal, an Undefined Name error results; if *global* and all other variables have values, a numerical result is returned from the calculation *global* − *expression*. This result has limited value. In general, execute ISOL with flag −3 clear.

Remarks: The result '$symb_2$' is an equation of the form '*global*=*expression*'. If *global* appears more than once, then '$symb_2$' is effectively the right side of an equation obtained by rearranging and solving '$symb_1$' to isolate the first occurrence of *global* on the left side of the equation.

If '$symb_1$' is an expression, it is treated as the left side of an equation '$symb_1$=0'.

If *global* appears in the argument of a function within '$symb_1$', that function must be an *analytic* function—a function for which the HP 48 provides an inverse. Thus ISOL cannot solve 'IP(X)=0' for X, since IP has no inverse.

Related Commands: COLCT, EXPAN, QUAD, SHOW

KERRM

Kermit Error Message Command: Returns the text of the most recent Kermit error packet.

Level 1	→	Level 1
	→	"*error-message*"

Keyboard Access: ◀ I/O NXT KERR

Affected by Flags: None

Remarks: If a Kermit transfer fails due to an error packet sent from the connected Kermit device to the HP 48, then executing KERRM retrieves and displays the error message. (Kermit errors not in packets are retrieved by ERRM rather than KERRM.)

Related Commands: FINISH, KGET, PKT, RECN, RECV, SEND, SERVER

KEY

Key Command: Returns to level 1 a test result and, if a key is pressed, returns to level 2 the row-column location $x_{n\,m}$ of that key.

Level 1	→	Level 2	Level 1
	→	$x_{n\,m}$	1
	→		0

Keyboard Access: PRG NXT IN KEY

Affected by Flags: None

Remarks: KEY returns a false result (0) to level 1 until a key is pressed. When a key is pressed, it returns a true result (1) to level

KEY

1 and $x_{n\,m}$ to level 2. The result $x_{n\,m}$ is a two-digit number that identifies the row and column location of the key just pressed. Unlike WAIT, which returns a three-digit number that identifies alpha and shifted keyboard planes, KEY returns the row-column location of *any* key pressed, including ◀, ▶, and α.

Example: The program « DO UNTIL KEY END 71 SAME » returns 1 to the stack if the ◀ key is pressed while the indefinite loop is running.

Related Commands: WAIT

KGET

Kermit Get Command: Used by a local Kermit to get a Kermit server to transmit the named object(s).

Level 1	→	Level 1
'*name*'	→	
"*name*"	→	
{ $name_{old}$ $name_{new}$ }	→	
{ $name_1$ ... $name_n$ }	→	
{{ $name_{old}$ $name_{new}$ } *name* ... }	→	

Keyboard Access: ◀ I/O SRVR KGET

Affected by Flags: I/O Device (−33), RECV Overwrite (−36), I/O Messages (−39)

I/O Data Format (−35) also affects KGET as follows:

- If the server is an HP 48, then the server's flag −35 affects KGET.
- If the server is not an HP 48 but the file being transferred originated from an HP 48, flag −35 has no effect.
- If the server is not an HP 48 and the file being transferred does not have the *%%HP...;* header, flag −35 tells the HP 48 whether to attempt parsing the incoming data.

Remarks: To rename an object when the local device gets it, include the old and new names in an embedded list. For example, { AAA BBB } KGET gets the variable named *AAA* but changes its name to *BBB*. { AAA BBB } CCC } KGET gets *AAA* as *BBB* and gets *CCC* under its own name. (If the original name is not legal on the HP 48, enter it as a string.)

Related Commands: BAUD, CKSM, FINISH, PARITY, RECN, RECV, SEND, SERVER, TRANSIO

KILL

Cancel Halted Programs Command: Cancels all currently halted programs. (Halted programs are typically canceled by pressing PRG NXT FUN KILL.) If KILL is executed within a program, that program is also canceled.

Keyboard Access: PRG NXT FUN KILL

Affected by Flags: None

Remarks: Canceled programs cannot be resumed.

KILL cancels *only* halted programs and the program from which KILL was executed, if any. Commands that halt programs are HALT and PROMPT.

Suspended programs cannot be canceled. Commands that suspend programs are INPUT and WAIT.

Related Commands: CONT, DOERR, HALT, PROMPT

LABEL

Label Axes Command: Labels axes in *PICT* with x- and y-axis variable names and with the minimum and maximum values of the display ranges.

Keyboard Access: ◀ PLOT NXT LABEL

Affected by Flags: None

Remarks: The horizontal axis name is chosen in the following priority order:

1. If the *axes* parameter in the reserved variable *PPAR* is a list, then the *x-axis* element from that list is used.
2. If *axes* parameter is not a list, then the independent variable name in *PPAR* is used.

The vertical axis name is chosen in the following priority order:

1. If the *axes* parameter in *PPAR* is a list, then the *y-axis* element from that list is used.
2. If *axes* is not a list, then the dependent variable name from PPAR is used.

Related Commands: AXES, DRAW, DRAX

LAST

Last Arguments Command: Returns copies of the arguments of the most recently executed command.

Keyboard Access: None. Must be typed in.

Remarks: LAST is provided for compatibility with the HP 28S. LAST is the same as LASTARG. See LASTARG.

LASTARG

Last Arguments Command: Returns copies of the arguments of the most recently executed command.

Level 1	→	Level n	...	Level 1
	→	obj_n	...	obj_1

Keyboard Access:

(→) ARG

PRG NXT ERROF LASTA

Affected by Flags: Last Arguments (−55)

Remarks: The objects return to the same stack levels that they originally occupied. Commands that take no arguments leave the current saved arguments unchanged.

When LASTARG follows a command that evaluates an algebraic expression or a program (as do ∂, $\int$, TAYLR, COLCT, DRAW, ROOT, ISOL, EVAL, and →NUM), the last arguments saved are from the evaluated algebraic expression or program, not from the original command.

Related Commands: LAST

LCD→

LCD to Graphics Object Command: Returns the current stack and menu display as a 131 × 64 graphics object.

Level 1	→	Level 1
	→	*grob*

LCD→

Keyboard Access: PRG GROB NXT LCD→

Affected by Flags: None

Example: LCD→ PICT STO PICTURE returns the current display to level 1 as a graphics object, stores it in *PICT*, then shows the image in the Picture environment.

Related Commands: →GROB, →LCD

→LCD

Graphics Object to LCD Command: Displays the graphics object from level 1, with its upper left pixel in the upper left corner of the display.

{ }

Level 1	→	Level 1
grob	→	

Keyboard Access: PRG GROB NXT →LCD

Affected by Flags: None

Remarks: If the graphics object is larger than 131 × 56, it is truncated.

Related Commands: BLANK, →GROB, LCD→

LIBEVAL

Evaluate Library Function Command: Evaluates unnamed library functions.

{}

Level 1	→	Level 1
$\#n_{function}$	→	

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Using LIBEVAL with random addresses can corrupt memory. $\#n_{function}$ is of the form *lllfffh*, where *lll* is the library number, and *fff* the function number.

Related Commands: EVAL, SYSEVAL

LIBS

Libraries Command: Lists the title, number, and port of each library attached to the current directory.

Level 1 →	Level 1
→	{ "*title*" n_{lib} n_{port} ... "*title*" n_{lib} n_{port} }

Keyboard Access: ← LIBRARY LIBS

Affected by Flags: None

Remarks: The title of a library often takes the form *LIBRARY-NAME* : *Description*. A library without a title is displayed as " ".

Related Commands: ATTACH, DETACH

LINE

Draw Line Command: Draws a line in *PICT* between the coordinates in levels 1 and 2.

Level 2	Level 1	→	Level 1
(x_1 , y_1)	(x_2 , y_2)	→	
{ # n_1 # m_1 }	{ # n_2 # m_2 }	→	

Keyboard Access: PRG PICT LINE

Affected by Flags: None

Example: This program:

```
« (0,0) (2,3) LINE { # 0d # 0d } PVIEW 7 FREEZE »
```

draws a line in *PICT* between two user-unit coordinates, displays *PICT* with pixel coordinate { # 0d # 0d } at the upper left corner of the picture display, and freezes the display.

Related Commands: ARC, BOX, TLINE

ΣLINE

Regression Model Formula Command: Returns an expression representing the best fit line according to the current statistical model, using X as the independent variable name, and explicit values of the slope and intercept taken from the reserved variable ΣPAR.

Level 1	→	Level 1
	→	'$symb_{formula}$'

Keyboard Access: ↶ STAT FIT ΣLINE

Affected by Flags: None

Remarks: For each curve fitting model, the following table indicates the form of the expression returned by ΣLINE, where m is the slope, x is the independent variable, and b is the intercept.

Model	Form of Expression
LINFIT	$mx + b$
LOGFIT	$m \ln(x) + \mathrm{b}$
EXPFIT	$b\mathrm{e}^{\mathrm{mx}}$
PWRFIT	bx^{m}

Example: If the current model is EXPFIT, and if the slope is 5 and the intercept 3, ΣLINE returns `'3*EXP(5*X)'`.

Related Commands: BESTFIT, COLΣ, CORR, COV, EXPFIT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, XCOL, YCOL

LINFIT

Linear Curve Fit Command: Stores LINFIT as the fifth parameter in the reserved variable ΣPAR, indicating that subsequent executions of LR are to use the linear curve fitting model.

Keyboard Access: ↩ STAT `ΣPAR MODL LINFI`

Affected by Flags: None

Remarks: LINFIT is the default specification in ΣPAR. For a description of ΣPAR, see appendix D, "Reserved Variables."

Related Commands: BESTFIT, EXPFIT, LOGFIT, LR, PWRFIT

LININ

Linear Test Function: Tests whether an algebraic is structurally linear for a given variable.

{ }

Level 2	Level 1	→	Level 1
'*symb*'	'*name*'	→	0/1

Keyboard Access: PRG TEST ◀ PREV LININ

Affected by Flags: None

Remarks: If any two subexpressions containing a variable (*name*) are combined only with addition and subtraction, and any subexpression containing the variable is at-most multiplied or divided by another factor not containing the variable, the algebraic ('*symb*') is determined to be linear for that variable.

LININ returns a 1 if the algebraic is linear for the variable, and a 0 if not.

Example:

```
'(X+1)*(Y^-2^Z)+(x/(3-Z^3)'
'X'
LININ
```

returns 1.

```
'(X^2-1)/(X+1)'
'X'
LININ
```

returns 0.

(Although this equation yields a linear equation when factored, LININ tests the equation as described above.)

LIST→

List to Stack Command: Takes a list of n objects and returns them to separate levels, and returns the total number of objects to level 1.

Level 1	→	Level n+1 ...	Level 2	Level 1
{ obj_1 ... obj_n }	→	obj_1 ...	obj_n	n

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: The command OBJ→ also provides this function. LIST→ is included for compatibility with the HP 28S.

Related Commands: ARRY→, DTAG, EQ→, →LIST, OBJ→, STR→

→LIST

Stack to List Command: Takes n objects from level n+1 through level 2 and returns a list of those n objects.

Level n+1 ...	Level 2	Level 1	→	Level 1
obj_1 ...	obj_n	n	→	{ obj_1 ... obj_n }

Keyboard Access: PRG TYPE →LIST

Affected by Flags: None

Example: The program

```
« DEPTH →LIST 'A' STO »
```

combines the entire contents of the stack into a list that is stored in variable A.

→LIST

Related Commands: →ARRY, LIST→, →STR, →TAG, →UNIT

ΣLIST

List Sum Command: Returns the sum of the elements in a list.

Level 1	→	Level 1
{ *list* }	→	*sum*

Keyboard Access: MTH LIST ΣLIST

Affected by Flags: None

Remarks: The elements in the list must be suitable for mutual addition.

Examples: { 5 8 2 } ΣLIST returns 15.

{ A B C 1 } ΣLIST returns 'A+B+C+1'.

Related Commands: ΠLIST, STREAM

ΔLIST

List Differences Command: Returns the first differences of the elements in a list.

Level 1	→	Level 1
{ *list* }	→	{ *differences* }

Keyboard Access: MTH LIST ΔLIST

Affected by Flags: None

Remarks: Adjacent elements in the list must be suitable for mutual subtraction.

Examples: { 4 20 1 17 60 91 } ΔLIST returns { 16 -19 16 43 31 }.

{ A B C 1 2 3 } ΔLIST returns { 'B-A' 'C-B' '1-C' 1 1 }.

{ 'A+3' 'X/5' 'Y^4' } ΔLIST returns
{ 'X/5-(A+3)' 'Y^4-X/5' }.

Related Commands: ΣLIST, ΠLIST, STREAM

ΠLIST

List Product Command: Returns the product of the elements in a list.

Level 1	→	Level 1
{ *list* }	→	*product*

Keyboard Access: PRG LIST ΠLIST

Affected by Flags: None

Remarks: The elements in the list must be suitable for mutual multiplication.

Examples: { 5 8 2 } ΠLIST returns 80.

{ A B C 1 } ΠLIST returns 'A*B*C'.

Related Commands: ΣLIST, ΔLIST, STREAM

LN

Natural Logarithm Analytic Function: Returns the natural (base e) logarithm of the argument.

{ }

Level 1	→	Level 1
z	→	ln z
'*symb*'	→	'LN(*symb*)'

Keyboard Access: ↷ LN

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Infinite Result Exception (−22)

Remarks: For x=0 or (0, 0), an `Infinite Result` exception occurs, or, if flag −22 is set, −MAXR is returned.

The inverse of EXP is a *relation*, not a function, since EXP sends more than one argument to the same result. The inverse relation for EXP is expressed by ISOL as the *general solution*

```
'LN(Z)+2*π*i*n1'
```

The function LN is the inverse of a *part* of EXP, a part defined by restricting the domain of EXP such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of EXP are called the *principal values* of the inverse relation. LN in its entirety is called the *principal branch* of the inverse relation, and the points sent by LN to the boundary of the restricted domain of EXP form the *branch cuts* of LN.

The principal branch used by the HP 48 for LN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued natural log function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

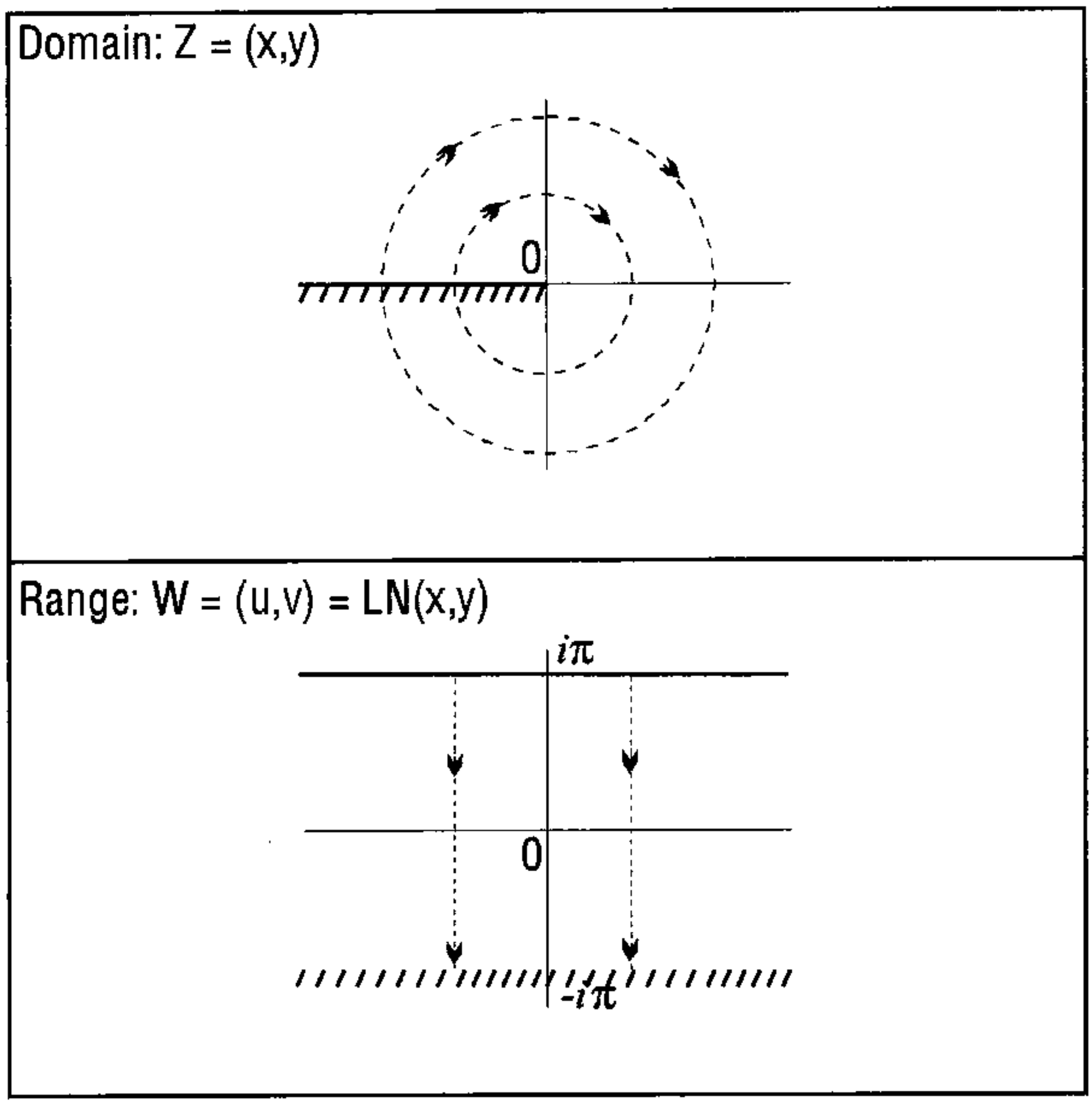
The graphs below show the domain and range of LN. The graph of the domain shows where the branch cut occurs: the heavy solid line marks

one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

These graphs show the inverse relation 'LN(Z)+2*π*i*n1' for the case *n1*=0. For other values of *n1*, the horizontal band in the lower graph is translated up (for *n1* positive) or down (for *n1* negative). Taken together, the bands cover the whole complex plane, which is the domain of EXP.

You can view these graphs with domain and range reversed to see how the domain of EXP is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (*x*, *y*). EXP sends this domain onto the whole complex plane in the range W = (*u*, *v*) = EXP(*x*, *y*) in the upper graph.

Related Commands: ALOG, EXP, ISOL, LNP1, LOG



LNP1

Natural Log of x Plus 1 Analytic Function: Returns $\ln(x+1)$.

{ }

Level 1	→	Level 1
x	→	ln(*x*+1)
'*symb*'	→	'LNP1(*symb*)'

Keyboard Access: (MTH) HYP (NXT) LNP1

Affected by Flags: Numerical Results (−3), Infinite Result Exception (−22)

Remarks: For values of x close to zero, 'LNP1(x)' returns a more accurate result than does 'LN(x+1)'. Using LNP1 allows both the argument and the result to be near zero, and it avoids an intermediate result near 1. The calculator can express numbers within 10^{-449} of zero, but within only 10^{-11} of 1.

For values of $x < -1$, an Undefined Result error results. For x=−1, an Infinite Result exception occurs, or, if flag −22 is set, LNP1 returns −MAXR.

Related Commands: EXPM, LN

LOG

Common Logarithm Analytic Function: Returns the common logarithm (base 10) of the argument.

{ }

Level 1	→	Level 1
z	→	log *z*
'*symb*'	→	'LOG(*symb*)'

Keyboard Access: ↱ LOG

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Infinite Result Exception (−22)

Remarks: For $x=0$ or (0, 0), an `Infinite Result` exception occurs, or, if flag −22 is set (no error), LOG returns −MAXR.

The inverse of ALOG is a *relation*, not a function, since ALOG sends more than one argument to the same result. The inverse relation for ALOG is expressed by ISOL as the *general solution*

```
'LOG(Z)+2*π*i*n1/2.3025850929'
```

The function LOG is the inverse of a *part* of ALOG, a part defined by restricting the domain of ALOG such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of ALOG are called the *principal values* of the inverse relation. LOG in its entirety is called the *principal branch* of the inverse relation, and the points sent by LOG to the boundary of the restricted domain of ALOG form the *branch cuts* of LOG.

The principal branch used by the HP 48 for LOG(z) was chosen because it is analytic in the regions where the arguments of the real-valued function are defined. The branch cut for the complex-valued LOG function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

You can determine the graph for LOG(z) from the graph for LN (see LN) and the relationship $\log z = \ln z \,/\, \ln 10$.

Related Commands: ALOG, EXP, ISOL, LN

LOGFIT

Logarithmic Curve Fit Command: Stores LOGFIT as the fifth parameter in the reserved variable ΣPAR, indicating that subsequent executions of LR are to use the logarithmic curve-fitting model.

Keyboard Access: ↰ STAT ΣPAR MODL LOGFI

Affected by Flags: None

Remarks: LINFIT is the default specification in ΣPAR. For a description of ΣPAR, see appendix D, "Reserved Variables."

Related Commands: BESTFIT, EXPFIT, LINFIT, LR, PWRFIT

LQ

LQ Factorization of a Matrix Command: Returns the LQ factorization of an $n \times m$ matrix.

{ }

Level 1	→	Level 3	Level 2	Level 1
[[*matrix*]] $_{A}$	→	[[*matrix*]] $_{L}$	[[*matrix*]] $_{Q}$	[[*matrix*]] $_{P}$

Keyboard Access: MTH MATR FACTR LQ

Affected by Flags: None

Remarks: LQ factors an $m \times n$ matrix A into three matrices:

- L is a lower $m \times n$ trapezoidal matrix.
- Q is an $n \times n$ orthogonal matrix.
- P is a $m \times m$ permutation matrix.

Where $P \times A = L \times Q$.

Related Commands: LSQ, QR

LR

Linear Regression Command: Uses the currently selected statistical model to calculate the linear regression coefficients (intercept and slope) for the selected dependent and independent variables in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 2	Level 1
	→	Intercept: x_1	Slope: x_2

Keyboard Access: ↰ STAT FIT LR

Affected by Flags: None

Remarks: The columns of independent and dependent data are specified by the first two elements in the reserved variable ΣPAR, set by XCOL and YCOL, respectively. (The default independent and dependent columns are 1 and 2.) The selected statistical model is the fifth element in ΣPAR. LR stores the intercept and slope (untagged) as the third and fourth elements, respectively, in ΣPAR.

The coefficients of the exponential (EXPFIT), logarithmic (LOGFIT), and power (PWRFIT) models are calculated using transformations that allow the data to be fitted by standard linear regression. The equations for these transformations appear in the table below, where b is the intercept and m is the slope. The logarithmic model requires positive x-values (XCOL), the exponential model requires positive y-values (YCOL), and the power model requires positive x- and y-values.

For a description of ΣPAR, see appendix D, "Reserved Variables."

Transformation Equations

Model	Transformation
Logarithmic	$y = b + m \ln(x)$
Exponential	$\ln(y) = \ln(b) + mx$
Power	$\ln(y) = \ln(b) + m \ln(x)$

LR

Related Commands: BESTFIT, COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, PREDX, PREDY, PWRFIT, XCOL, YCOL

LSQ

Least Squares Solution Command: Returns the minimum norm least squares solution to any system of linear equations where $A \times X = B$.

{}

Level 2	Level 1	→	Level 1
[*array*]$_B$	[[*matrix*]]$_A$	→	[*array*]$_X$
[[*matrix*]]$_B$	[[*matrix*]]$_A$	→	[[*matrix*]]$_X$

Keyboard Access:

(↩) SOLVE 8'S LSQ

MTH MATR LSQ

Affected by Flags: Singular Values (−54)

Remarks: If B is a vector, the resulting vector has a minimum Euclidean norm $||X||$ over all vector solutions that minimize the residual Euclidean norm $||A \times X - B||$. If B is a matrix, each column of the resulting matrix, X_i, has a minimum Euclidean norm $||X_i||$ over all vector solutions that minimize the residual Euclidean norm $||A \times X_i - B_i||$.

If A has less than full row rank (the system of equations is underdetermined), an infinite number of solutions exist. LSQ returns the solution with the minimum Euclidean length.

If A has less than full column rank (the system of equations is overdetermined), a solution that satisfies all the equations may not exist. LSQ returns the solution with the minimum residuals of $A \times X - B$.

Related Commands: LQ, RANK, QR, /

LU

LU Decomposition of a Square Matrix Command: Returns the LU decomposition of a square matrix.

{}

Level 1	→	Level 3	Level 2	Level 1
[[*matrix*]]$_A$	→	[[*matrix*]]$_L$	[[*matrix*]]$_U$	[[*matrix*]]$_P$

Keyboard Access: (MTH) MATR FACTR LU

Affected by Flags: None

Remarks: When solving an exactly determined system of equations, inverting a square matrix, or computing the determinant of a matrix, the HP 48 factors a square matrix into its Crout LU decomposition using partial pivoting.

The Crout LU decomposition of A is a lower-triangular matrix L, an upper-triangular matrix U with ones on its diagonal, and a permutation matrix P, such that $P \times A = L \times U$. The results satisfy $P \times A \cong L \times U$.

Related Commands: DET, INV, LSQ, /

MANT

Mantissa Function: Returns the mantissa of the argument.

{}

Level 1	→	Level 1
x	→	y_{mant}
'*symb*'	→	'MANT(*symb*)'

Keyboard Access: (MTH) REAL (NXT) MANT

MANT

Affected by Flags: Numerical Results (−3)

Example: −1.2E34 MANT returns 1.2.

Related Commands: SIGN, XPON

↑MATCH

Bottom-Up Match and Replace Command: Rewrites an expression.

Level 2	Level 1	→	Level 2	Level 1
'$symb_1$'	{ '$symb_{pat}$' '$symb_{repl}$' }	→	'$symb_2$'	0/1
'$symb_1$'	{ '$symb_{pat}$' '$symb_{repl}$' '$symb_{cond}$' }	→	'$symb_2$'	0/1

Keyboard Access: ← SYMBOLIC NXT ↑MATCH

Affected by Flags: None

Remarks: ↑MATCH rewrites expressions or subexpressions that match a specified pattern '$symb_{pat}$'. An optional condition, '$symb_{cond}$', can further restrict whether a rewrite occurs. A test result is also returned to indicate if command execution produced a rewrite; 1 if it did, 0 if it did not.

The pattern '$symb_{pat}$' and replacement '$symb_{repl}$' can be normal expressions; for example, you can replace 'SIN(π/6)' with '1/2'. You can also use a "wildcard" in the pattern (to match any subexpression) and in the replacement (to represent that expression). A wildcard is a name that begins with &, such as the name '&A', used in replacing 'SIN(&A+π)' with '−SIN(&A)'. Multiple occurrences of a particular wildcard in a pattern must match identical subexpressions.

↑MATCH works from bottom up; that is, it checks the lowest level (most deeply nested) subexpressions first. This approach works well for simplification. A subexpression simplified during one execution of ↑MATCH will be a simpler argument of its parent expression, so the parent expression can be simplified by another execution of ↑MATCH.

Several subexpressions can be simplified by one execution of ↑MATCH provided none is a subexpression of any other.

Examples: This sequence:

```
'SIN(π/6)' { 'SIN(π/6) '1/2' } ↑MATCH
```

returns '1/2' to level 2 and 1 (indicating a replacement was made) to level 1.

This sequence:

```
'SIN(X+π)' { 'SIN(&A+π)' '-SIN(&A)' } ↑MATCH
```

returns '-SIN(X)' to level 2 and 1 to level 1.

This sequence:

```
'W+√(SQ(5))' { '√(SQ&A))' '&A' '&A≥0' } ↑MATCH
```

returns 'W+5' to level 2 and 1 to level 1.

Related Commands: ↓MATCH

↓MATCH

Match Pattern Down Command: Rewrites an expression.

Level 2	Level 1	→	Level 2	Level 1
'$symb_1$'	{ '$symb_{pat}$' '$symb_{repl}$' }	→	'$symb_2$'	0/1
'$symb_1$'	{ '$symb_{pat}$' '$symb_{repl}$' '$symb_{cond}$' }	→	'$symb_2$'	0/1

Keyboard Access: ← SYMBOLIC NXT ↓MAT

Affected by Flags: None

Remarks: ↓MATCH rewrites expressions or subexpressions that match a specified pattern '$symb_{pat}$'. An optional condition, '$symb_{cond}$', can further restrict whether a rewrite occurs. A test result is also returned to indicate if command execution produced a rewrite; 1 if it did, 0 if it did not.

↓MATCH

The pattern '$symb_{pat}$' and replacement '$symb_{repl}$' can be normal expressions; for example, you can replace .5 with 'SIN(π/6'). You can also use a "wildcard" in the pattern (to match any subexpression) and in the replacement (to represent that expression). A wildcard is a name that begins with &, such as the name '&A', used in replacing 'SIN(&A+&B)' with 'SIN(&A)*COS(&B)+COS(&A)*SIN(&B)'. Multiple occurrences of a particular wildcard in a pattern must match identical subexpressions.

↓MATCH works from top down; that is, it checks the entire expression first. This approach works well for expansion. An expression expanded during one execution of ↓MATCH will contain additional subexpressions, and those subexpressions can be expanded by another execution of ↓MATCH. Several expressions can be expanded by one execution of ↓MATCH provided none is a subexpression of any other.

Examples:

```
.5 ( .5 'SIN(π/6)' ) ↓MATCH
```

returns 'SIN(π/6)' to level 2 and 1 to level 1.

```
'SIN(U+V)' ( 'SIN(&A+&B)'
'SIN(&A)*COS(&B)+COS(&A)*SIN(&B)' ) ↓MATCH
```

returns 'SIN(U)*COS(V)+COS(U)*SIN(V)' to level 2 and 1 to level 1.

This sequence:

```
'SIN(5*Z)' ( 'SIN(&A+&B)'
'Σ(K=0,&A,COMB(&A,K)*SIN(K*π)*
COS(&B^(&A-K)*SIN(&B)^K)'
'ABS(IP(&A))==&A' ) ↓MATCH
```

returns

```
'Σ(K=0,5,COMB(5,K)*SIN(K*π)*COS(Z)^(5-K)*SIN(Z)^K)'
```

to level 2 and 1 to level 1.

Related Commands: ↑MATCH

MAX

Maximum Function: Returns the greater (more positive) of the arguments.

{}

Level 2	Level 1	→	Level 1
x	y	→	max(x, y)
x	'$symb$'	→	'MAX(x, $symb$)'
'$symb$'	x	→	'MAX($symb$, x)'
'$symb_1$'	'$symb_2$'	→	'MAX($symb_1$, $symb_2$)'
x_unit_1	y_unit_2	→	max(x_unit_1, y_unit_2)

Keyboard Access: (MTH) REAL MAX

Affected by Flags: Numerical Results (−3)

Examples: 10 -23 MAX returns 10.

-10 -23 MAX returns -10.

1_m 9_cm MAX returns 1_m.

Related Commands: MIN

MAXR

Maximum Real Function: Returns the symbolic constant 'MAXR' or its numerical representation, 9.999999999999E499.

Level 1	→	Level 1
	→	'MAXR'
	→	9.999999999999E499

Keyboard Access: (MTH) (NXT) CONS (NXT) MAXR

MAXR

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

MAXR returns its numerical representation if flag −2 or −3 is set; otherwise, it returns its symbolic representation.

Remarks: MAXR is the largest numerical value that can be represented by the HP 48.

Related Commands: e, i, MINR, π

MAXΣ

Maximum Sigma Command: Finds the maximum coordinate value in each of the m columns of the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{max}
	→	$[x_{max1}\ x_{max2}\ \cdots\ x_{maxm}]$

Keyboard Access: ↩ STAT 1 VAR MAXΣ

Affected by Flags: None

Remarks: The maxima are returned as a vector of m real numbers, or as a single real number if $m = 1$.

Related Commands: BINS, MEAN, MINΣ, SDEV, TOT, VAR

MCALC

Make Calculated Value Command: Designates a variable as a calculated value (not user-defined) for the Multiple-Equation Solver.

Level 1	→	Level 1
'name'	→	
{ *list* }	→	
" *ALL*"	→	

Keyboard Access: ◀ EQ LIB MES MCAL

Affected by Flags: None

Remarks: MCALC designates a single variable, a list of variables, or all variables as calculated values.

Related Commands: MUSER

MEAN

Mean Command: Returns the mean of each of the m columns of coordinate values in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{mean}
	→	$[\ x_{mean1}\ x_{mean2}\ \cdots\ x_{meanm}\ ]$

Keyboard Access: ◀ STAT 1VAR MEAN

Affected by Flags: None

MEAN

Remarks: The mean is returned as a vector of m real numbers, or as a single real number if $m = 1$. The mean is computed from the formula:

$$\frac{1}{n}\sum_{i=1}^{n} x_i$$

where x_i is the ith coordinate value in a column, and n is the number of data points.

Related Commands: BINS, MAXΣ, MINΣ, SDEV, TOT, VAR

MEM

Memory Available Command: Returns the number of bytes of available RAM.

Level 1	→	Level 1
	→	*x*

Keyboard Access: ◀ MEMORY MEM

Affected by Flags: None

Remarks: The number returned is only a rough indicator of usable available memory, since recovery features (LASTARG, ▶ UNDO, and ▶ CMD) consume or release varying amounts of memory with each operation.

Before it can assess the amount of memory available, MEM must remove objects in temporary memory that are no longer being used. This clean-up process (also called "garbage collection") also occurs automatically at other times when memory is full. Since this process can slow down calculator operation at undesired times, you can force it to occur at a desired time by executing MEM. In a program, execute MEM DROP.

Related Commands: BYTES

MENU

Display Menu Command: Displays a built-in menu or a library menu, or defines and displays a custom menu.

Level 1	→	Level 1
x_{menu}	→	
{ $list_{definition}$ }	→	
'$name_{definition}$'	→	
obj	→	

Affected by Flags: None

Remarks: A built-in menu is specified by a real number x_{menu}. The format of x_{menu} is *mm.pp*, where *mm* is the menu number and *pp* is the page of the menu. If *pp* doesn't correspond to a page of the specified menu, the first page is displayed. The following table lists the HP 48 built-in menus and the corresponding menu numbers.

Menu #	Menu Name
0	Last Menu
1	CST
2	VAR
3	MTH
4	MTH VECTR
5	MTH MATR
6	MTH MATR MAKE
7	MTH MATR NORM
8	MTH MATR FACTR
9	MTH MATR COL
10	MTH MATR ROW
11	MTH LIST
12	MTH HYP
13	MTH PROB
14	MTH REAL

Menu #	Menu Name
15	MTH BASE
16	MTH BASE LOGIC
17	MTH BASE BIT
18	MATH BASE BYTE
19	MTH FFT
20	MTH CMPL
21	MTH CONS
22	PRG
23	PRG BRCH
24	PRG BRCH IF
25	PRG BRCH CASE
26	PRG BRCH START
27	PRG BRCH FOR
28	EDIT
29	PRG BRCH DO

MENU

Menu #	Menu Name
30	SOLVE ROOT SOLVR
31	PRG BRCH WHILE
32	PRG TEST
33	PRG TYPE
34	PRG LIST
35	PRG LIST ELEM
36	PRG LIST PROC
37	PRG GROB
38	PRG PICT
39	PRG IN
40	PRG OUT
41	PRG RUN
42	UNITS (Units Catalog Menu)
43	UNITS LENG
44	UNITS AREA
45	UNITS VOL
46	UNITS TIME
47	UNITS SPEED
48	UNITS MASS
49	UNITS FORCE
50	UNITS ENRG
51	UNITS POWR
52	UNITS PRESS
53	UNITS TEMP
54	UNITS ELEC
55	UNITS ANGL
56	UNITS LIGHT
57	UNITS RAD
58	UNITS VISC
59	UNITS
60	PRG ERROR IFERR
61	PRG ERROR

Menu #	Menu Name
62	CHARS
63	MODES
64	MODES FMT
65	MODES ANGL
66	MODES FLAG
67	MODES KEYS
68	MODES MENU
69	MODES MISC
70	MEMORY
71	MEMORY DIR
72	MEMORY ARITH
73	STACK
74	SOLVE
75	SOLVE ROOT
76	SOLVE DIFFE
77	SOLVE POLY
78	SOLVE SYS
79	SOLVE TVM
80	SOLVE TVM SOLVR
81	PLOT
82	PLOT PTYPE
83	PLOT PPAR
84	PLOT 3D
85	PLOT 3D PTYPE
86	PLOT 3D VPAR
87	PLOT STAT
88	PLOT STAT PTYPE
89	PLOT STAT ΣPAR
90	PLOT STAT ΣPAR MODL
91	PLOT STAT DATA
92	PLOT FLAG
93	SYMBOLIC

Menu #	Menu Name
94	TIME
95	TIME ALARM
96	STAT
97	STAT DATA
98	STAT ΣPAR
99	STAT ΣPAR MODL
100	STAT 1VAR
101	STAT PLOT
102	STAT FIT
103	STAT SUMS
104	IO
105	IO SRVR
106	IO IOPAR

Menu #	Menu Name
107	IO PRINT
108	IO PRINT PRTPA
109	IO SERIA
110	LIBRARY
111	LIBRARY PORTS
112	LIBRARY PORTS :0:
113	EQ LIB
114	EQ LIB EQLIB
115	EQ LIB COLIB
116	EQ LIB MES
117	EQ LIB UTILS

Library menus are specified in the same way as built-in menus, with the library number serving as the menu number.

Custom menus are specified by a list of the form { "*label-object*" *action-object* } (see appendix D, "Reserved Variables," for details) or a name containing a list ('$name_{\texttt{definition}}$'). Either argument is stored in reserved variable *CST*, and the custom menu is subsequently displayed.

MENU takes *any* object as a valid argument and stores it in *CST*. However, the calculator can build a custom menu *only* if *CST* contains a list or a name containing a list. Thus, if an object other than a list or name containing a list is supplied to MENU, a `Bad Argument Type` error will occur when the calculator attempts to display the custom menu.

Examples: `5 MENU` displays the first page of the MTH MATR NORM menu.

`48.02 MENU` displays the second page of the UNITS MASS menu.

`{ A 123 "ABC" } MENU` displays the custom menu defined by the list argument.

`'MYMENU' MENU` displays the custom menu defined by the name argument.

MENU

Related Commands: RCLMENU, TMENU

MERGE

Merge RAM Card Command: Merges the RAM from the card in port 1 with the rest of main user memory. Merged memory is no longer independent.

Level 1	→	Level 1
1	→	

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: MERGE is provided for compatibility with the HP 48S series. See MERGE1.

Related Commands: FREE1, MERGE1

MERGE1

Merge RAM Card Command: Merges the RAM from the card in port 1 with the rest of main user memory. Merged memory is no longer independent.

Keyboard Access: ← LIBRARY MERG

Affected by Flags: None

Remarks: If the RAM card contains library or backup objects, they are moved to port 0 before the RAM is merged. Library and backup objects can exist only in independent memory (port 1 through 33 unmerged, or port 0).

Cards larger than 128K cannot be merged, and cannot be plugged into port 1.

Related Commands: FREE, FREE1, MERGE

MIN

Minimum Function: Returns the lesser (more negative) of its two arguments.

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	min(*x*, *y*)
x	'*symb*'	→	'MIN(*x*, *symb*)'
'*symb*'	*x*	→	'MIN(*symb*, *x*)'
'$symb_1$'	'$symb_2$'	→	'MIN($symb_1$, $symb_2$)'
x_unit_1	y_unit_2	→	min(x_unit_1, y_unit_2)

Keyboard Access: MTH REAL MIN

Affected by Flags: Numerical Results (−3)

Examples: 10 23 MIN returns 10.

-10 -23 MIN returns -23.

1_m 9_cm MIN returns 9_cm.

Related Commands: MAX

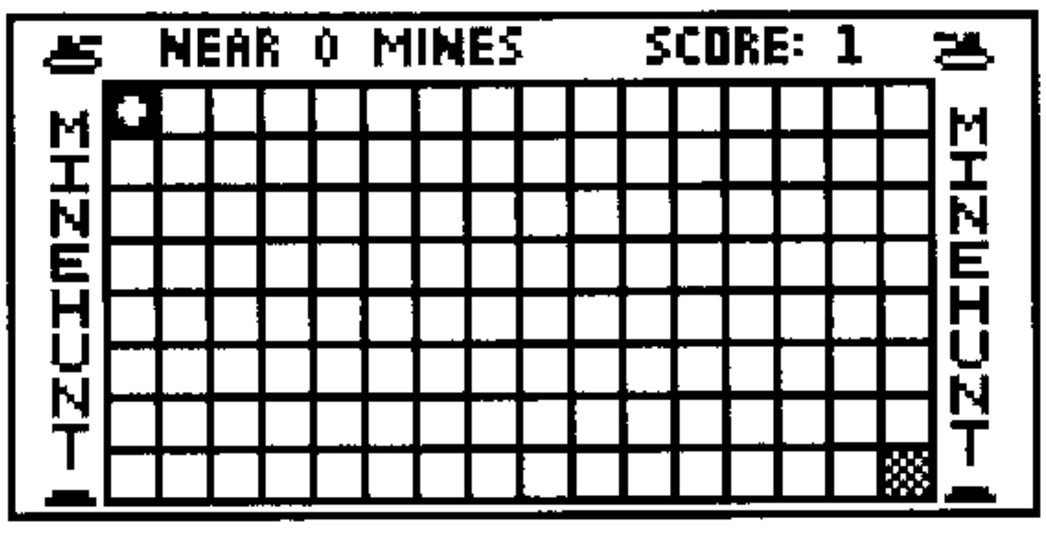
MINEHUNT

MINEHUNT Game Command: Starts the MINEHUNT game.

Keyboard Access: ◀ EQ LIB UTILS MINE

Affected by Flags: None

Remarks: In the game, you are standing in the upper-left corner of an 8 × 16 battlefield grid. Your mission is to travel safely to the lower-right corner, avoiding invisible mines along the way. The game tells you how many mines are under the eight squares adjacent to your position.



Use the number or arrow keys to cross the battlefield one square at a time. (Use 7, 9, 1, and 3 to move diagonally.) You can exit the game any time by pressing CANCEL.

To interrupt and save a game, press STO. This creates a variable *MHpar* in the current directory and ends the game. If *MHpar* exists when you next start Minehunt, the interrupted game resumes and *MHpar* is purged.

You can change the number of mines in the battlefield by creating a variable named *Nmines* containing the desired number. *Nmines* must contain a real number (1 to 64). If *Nmines* is negative, the mines are visible during the game.

MINIT

Multiple Equation Menu Initialization Command: Creates the reserved variable *Mpar*.

Keyboard Access: ◀ EQ LIB MES MINIT

Affected by Flags: None

Remarks: MINIT takes multiple equations stored in *EQ* and creates the multiple equation reserved variable *Mpar*. See appendix D, "Reserved Variables," for information about *Mpar*.

Related Commands: MITM, MROOT, MSOLVR

MINR

Minimum Real Function: Returns the symbolic constant 'MINR' or its numerical representation, 1.0000000000000E-499.

Level 1	→	Level 1
	→	'MINR'
	→	1.0000000000E-499

Keyboard Access: MTH NXT CONS NXT MINR

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

MAXR returns its numerical representation if flag −2 or −3 is set; otherwise, it returns its symbolic representation.

Remarks: MINR is the smallest nonzero numerical value that can be represented by the HP 48.

Related Commands: e, i, MAXR, π

MINΣ

Minimum Sigma Command: Finds the minimum coordinate value in each of the m columns of the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{min}
	→	$[\ x_{min1}\ x_{min2}\ \cdots\ x_{minm}\ ]$

Keyboard Access: ◀ STAT 1 VAR MINΣ

Affected by Flags: None

Remarks: The minima are returned as a vector of m real numbers, or as a single real number if $m = 1$.

Related Commands: BINS, MAXΣ, MEAN, SDEV, TOT, VAR

MITM

Multiple Equation Menu Item Order Command: Changes multiple equation menu titles and order.

Level 2	Level 1	→	Level 1
"*title*"	{ *list* }	→	

Keyboard Access: ◀ EQ LIB MES MITM

Affected by Flags: None

Remarks: *list* contains the variable names in the order you want. Use "" to indicate a blank label. You must include *all* variables in the original menu and no others.

Related Commands: MINIT

MOD

Modulo Function: Returns a remainder defined by:

$$x \bmod y = x - y \text{ floor } (x/y)$$

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	*x* mod *y*
x	'*symb*'	→	'MOD(*x*, *symb*)'
'*symb*'	*x*	→	'MOD(*symb*, *x*)'
'$symb_1$'	'$symb_2$'	→	'MOD($symb_1$, $symb_2$)'

Keyboard Access: (MTH) REAL MOD

Affected by Flags: Numerical Results (−3)

Remarks: Mod (x, y) is periodic in x with period y. Mod (x, y) lies in the interval [0, y) for $y > 0$ and in (y, 0] for $y < 0$.

Related Commands: FLOOR, /

MROOT

Multiple Roots Command: Uses the Multiple-Equation Solver to solve for one or more variables using the equation set in *Mpar*.

{ }

Level 1	→	Level 1
'*name*'	→	*x*
"*ALL*"	→	

MROOT

Keyboard Access: ◀ EQ LIB MES MROO

Affected by Flags: None

Remarks: Solves for one or more variables starting with only user-defined values, and leaves found values in the variables. No status messages are displayed. Given a variable name, MROOT returns the found value; it can also take "FILL" (stores a found value for each variable) and return nothing to the stack.

Related Commands: MCALC, MUSER

MSGBOX

Message Box Command: Creates a user-defined message box.

{ }

Level 1	→	Level 1
"*message*"	→	

Keyboard Access: PRG NXT OUT MSGB

Affected by Flags: None

Remarks: MSGBOX displays "*message*" in the form of a standard message box. Message text longer than 75 characters (including spaces) is truncated to 75 characters. You can use spaces and new-line characters (▶ ↵) to control word-wrapping and line breaks within the message.

Program execution resumes when the message box is exited by selecting OK or CANCL.

Related Commands: CHOOSE, INFORM, PROMPT

MSOLVR

Multiple-Equation Solver Command: Gets the Multiple-Equation Solver variable menu for the set of equations defined by *Mpar*.

Keyboard Access:

(←) EQ LIB MES MSOL

(←) EQ LIB EQLIB MSOL

Affected by Flags: None

Remarks: The Multiple-Equation Solver application can solve a set of two or more equations for unknown variables by finding the roots of each equation, one at a time.

The Multiple-Equation Solver uses the list of equations stored in *EQ*. "Equations" in this context includes programs, expressions, and variable names that evaluate to a single value. The Multiple-Equation Solver requires that *EQ* contain more than one equation—that is, the HP Solve application would include the NUM menu label for *EQ*. The solver uses *EQ* to create a reserved variable *Mpar* that is used during the solution process. *Mpar* contains the equation set plus additional information. See appendix D, "Reserved Variables," for information about *Mpar*.

Related Commands: EQNLIB, SOLVEQN

MUSER

Make User-Defined Variable Command: Designates a variable as user-defined for the Multiple-Equation Solver.

Level 1	→	Level 1
'name'	→	
{ *list* }	→	
"*ALL*"	→	

MUSER

Keyboard Access: ◀ EQ LIB MES MUSE

Affected by Flags: None

Remarks: MUSER designates a single variable, a list of variables, or all variables as user-defined.

Related Commands: MCALC

NDIST

Normal Distribution Command: Returns the normal probability distribution (bell curve) at x based on the mean m and variance v of the normal distribution.

{ }

Level 3	Level 2	Level 1	→	Level 1
m	*v*	*x*	→	*ndist(m,v,x)*

Keyboard Access: MTH NXT PROB NXT NDIST

Affected by Flags: None

Remarks: NDIST is calcluated using this formula:

$$ndist(m, v, x) = \frac{e^{-\frac{(x-m)^2}{2v}}}{\sqrt{2\pi v}}$$

Related Commands: UTPN

NEG

Negate Analytic Function: Changes the sign or negates an object.

{ }

Level 1	→	Level 1
z	→	$-z$
$\#n_1$	→	$\#n_2$
[*array*]	→	[$-$*array*]
'*symb*'	→	'$-$(*symb*)'
x_unit	→	$-$*x_unit*
$grob_1$	→	$grob_2$
$PICT_1$	→	$PICT_2$

Keyboard Access:

+/–

MTH NXT CMPL NXT NEG

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: Negating an array creates a new array containing the negative of each of the original elements. Negating a binary number takes its two's complement (complements each bit and adds 1).

Negating a graphics object "inverts" it (toggles each pixel from on to off, or vice-versa). If the argument is *PICT*, the graphics object stored in *PICT* is inverted.

Related Commands: ABS, CONJ, NOT, SIGN

NEWOB

New Object Command: Creates a new copy of the specified object.

Level 1	→	Level 1
obj	→	*obj*

Keyboard Access: ◀ MEMORY NEWO

Affected by Flags: Last Arguments (−55)

In order for NEWOB to immediately release the memory occupied by the original copy, flag −55 must be set so that the copy is not saved as a last argument.

Remarks: NEWOB has two main uses:

- NEWOB enables the purging of a library or backup object that has been recalled from a port. NEWOB creates a new, separate copy of the object in memory, thereby allowing the original copy to be purged.
- Creating a new copy of an object that originated in a larger composite object (such as a list) allows you to recover the memory associated with the larger object when that larger object is no longer needed.

Examples: `:0:BKUP1 RCL NEWOB :0:BKUP1 PURGE` recalls and purges the backup object *BKUP1*.

`3 GET NEWOB` retrieves the third element out of a list in the stack, recovering the memory occupied by the whole list.

Related Commands: MEM, PURGE

NEXT

NEXT Command: Ends definite loop structures.

See the FOR and START command entries for syntax information.

Keyboard Access:

PRG BRCH START BRCH

PRG BRCH FOR NEXT

Remarks: See the FOR and START keyword entries for more information.

Related Commands: FOR, START, STEP

NEXT

Next Operation: Returns but does not execute the next one or two steps of a program.

Keyboard Access: PRG NXT RUN NEXT

Affected by Flags: None

Related Commands: SST, SST↓

NOT

NOT Command: Returns the one's complement or logical inverse of the argument.

NOT

{ }

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$
T/F	→	0/1
"*string*$_1$"	→	"*string*$_2$"
'*symb*'	→	'NOT *symb*'

Keyboard Access:

PRG TEST NXT NOT

MTH BASE NXT LOGIC NOT

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: When the argument is a binary integer or string, NOT complements each bit in the argument to produce the result.

- A binary integer is treated as a sequence of bits as long as the current wordsize.
- A string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code).

When the argument is a real number or symbolic, NOT does a true/false test. The result is 1 (true) if the argument is zero; it is 0 (false) if the argument is nonzero. This test is usually done on a test result (T/F).

If the argument is an algebraic object, then the result is an algebraic of the form 'NOT *symb*'. Execute →NUM (or set flag −3 before executing NOT) to produce a numeric result from the algebraic result.

Related Commands: AND, OR, XOR

NOVAL

INFORM Place Holder/Result Command: Place holder for reset and initial values in user-defined dialog boxes. NOVAL is returned to the stack when a field is empty.

Keyboard Access: (PRG) (NXT) IN NOVA

Affected by Flags: None

Remarks: NOVAL is used to mark an empty field in a user-defined dialog box created with the INFORM command. INFORM defines fields sequentially. If default values are used for those fields, the defaults must be defined in the same order as the fields were defined. To skip over (not provide defaults for) some of the fields, use the NOVAL command.

After INFORM terminates, NOVAL is returned to the stack (on level 2) if a field is empty and OK or (ENTER) is selected.

Related Commands: INFORM

NSUB

Number of Sublist Command: Provides a way to access the current sublist position during an iteration of a program or command applied using DOSUBS.

Keyboard Access: (PRG) LIST PROC NSUB

Affected by Flags: None

Remarks: Returns an `Undefined Local Name` error if executed when DOSUBS is not active.

Related Commands: DOSUBS, ENDSUB

NUM

Character Number Command: Returns the character code *n* for the first character in the string.

{ }

Level 1	→	Level 1
" string"	→	*n*

Keyboard Access:

◀ CHARS NUM

PRG TYPE NXT NUM

Affected by Flags: None

Remarks: The character codes are an extension of ISO 8859/1. Codes 128 through 159 are unique to the HP 48.

The following tables show the relation between character codes (results of NUM, arguments to CHR) and characters (results of CHR, arguments to NUM).

Character Codes (0 — 127)

NUM	CHR	NUM	CHR	NUM	CHR	NUM	CHR
0	■	32		64	@	96	`
1	■	33	!	65	A	97	a
2	■	34	"	66	B	98	b
3	■	35	#	67	C	99	c
4	■	36	$	68	D	100	d
5	■	37	%	69	E	101	e
6	■	38	&	70	F	102	f
7	■	39	'	71	G	103	g
8	■	40	(	72	H	104	h
9	■	41	)	73	I	105	i
10	■	42	*	74	J	106	j
11	■	43	+	75	K	107	k
12	■	44	,	76	L	108	l
13	■	45	-	77	M	109	m
14	■	46	.	78	N	110	n
15	■	47	/	79	O	111	o
16	■	48	0	80	P	112	p
17	■	49	1	81	Q	113	q
18	■	50	2	82	R	114	r
19	■	51	3	83	S	115	s
20	■	52	4	84	T	116	t
21	■	53	5	85	U	117	u
22	■	54	6	86	V	118	v
23	■	55	7	87	W	119	w
24	■	56	8	88	X	120	x
25	■	57	9	89	Y	121	y
26	■	58	:	90	Z	122	z
27	■	59	;	91	[	123	{
28	■	60	<	92	\	124	|
29	■	61	=	93	]	125	}
30	■	62	>	94	^	126	~
31	■	63	?	95	_	127	■

Character Codes (128 — 255)

NUM	CHR	NUM	CHR	NUM	CHR	NUM	CHR
128	Ä	160		192	┴	224	Ω
129	Å	161	¡	193	┬	225	ß
130	Ç	162	¢	194	├	226	Γ
131	É	163	£	195	─	227	π
132	Ñ	164	¤	196	┼	228	Σ
133	Ö	165	¥	197	╞	229	σ
134	Ü	166	¦	198	╟	230	µ
135	á	167	§	199	╚	231	τ
136	à	168	¨	200	╔	232	Φ
137	â	169	©	201	╩	233	Θ
138	ä	170	ª	202	╦	234	Ω
139	ã	171	«	203	╠	235	δ
140	å	172	¬	204	═	236	∞
141	ç	173		205	╬	237	φ
142	é	174	®	206	╧	238	ε
143	è	175	¯	207	╨	239	∩
144	ê	176	°	208	╤	240	≡
145	ë	177	±	209	╥	241	±
146	í	178	²	210	╙	242	≥
147	ì	179	³	211	╘	243	≤
148	î	180	´	212	╒	244	⌠
149	ï	181	µ	213	╓	245	⌡
150	ñ	182	¶	214	╫	246	÷
151	ó	183	·	215	╪	247	≈
152	ò	184	¸	216	┘	248	°
153	ô	185	¹	217	┌	249	∙
154	ö	186	º	218	█	250	·
155	õ	187	»	219	▄	251	√
156	ú	188	¼	220	▌	252	ⁿ
157	ù	189	½	221	▐	253	²
158	û	190	¾	222	▀	254	■
159	ü	191	¿	223	α	255		

Related Commands: CHR, POS, REPL, SIZE, SUB

→NUM

Evaluate to Number Command: Evaluates a symbolic argument object (other than a list) and returns the numerical result.

{ }

Level 1	→	Level 1
obj_{symb}	→	z

Keyboard Access: ◀ →NUM

Affected by Flags: None

Remarks: →NUM repeatedly evaluates a symbolic argument until a numerical result is achieved. The effect is the same as evaluating the symbolic argument in Numerical Result mode (flag −3 set).

Related Commands: EVAL, SYSEVAL

NUMX

Number of X-Steps Command: Sets the number of x-steps for each y-step in 3D perspective plots.

{ }

Level 1	→	Level 1
n_x	→	

Keyboard Access: ◀ PLOT NXT 3D VPAR NXT NUMX

Affected by Flags: None

Remarks: The number of x-steps is the number of independent variable points plotted for each dependent variable point plotted. This number must be 2 or more. This value is stored in the reserved

NUMX

variable VPAR. YSLICE is the only 3D plot type that does not use this value.

Related Commands: NUMY

NUMY

Number of Y-Steps Command: Sets the number of y-steps across the view volume in 3D perspective plots.

{ }

Level 1	→	Level 1
n_y	→	

Keyboard Access: ◀ PLOT NXT 3D VPAR NXT NUMY

Affected by Flags: None

Remarks: The number of y-steps is the number of dependent variable points plotted across the view volume. This number must be 2 or more. This value is stored in the reserved variable VPAR.

Related Commands: NUMX

NΣ

Number of Rows Command: Returns the number of rows in the current statistical matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	n_{rows}

Keyboard Access: ◀ STAT SUMS NΣ

Affected by Flags: None

Related Commands: ΣX, ΣX*Y, ΣX^2, ΣY, ΣY^2

OBJ→

Object to Stack Command: Separates an object into its components onto the stack. For some object types, the *number* of components is returned to level 1.

Level 1	→	Level n+1 ...	Level 2	Level 1
(x,y)	→		x	y
$\{\ obj_1\ \ \ldots\ \ obj_n\ \}$	→	obj_1	obj_n	n
$[\ x_1\ \ \ldots\ \ x_n\ ]$	→	x_1	x_n	$\{\ n\ \}$
$[[\ x_{1\,1}\ \ \ldots\ \ x_{m\,n}\ ]]$	→	$x_{1\,1}$	$x_{m\,n}$	$\{\ m\ n\ \}$
"*obj*"	→			*evaluated-object*
'*symb*'	→	arg_1 ... arg_n	n	'*function*'
x_unit	→		x	1_*unit*
:*tag*:*obj*	→		*obj*	"*tag*"

Keyboard Access:

◀ CHARS NXT OBJ→

PRG TYPE OBJ→

Affected by Flags: None

Remarks: If the argument is a complex number, list, array, or string, OBJ→ provides the same functions as C→R, LIST→, ARRY→, and STR→, respectively. For lists, OBJ→ also returns the number of list elements. If the argument is an array, OBJ→ also returns the dimensions $\{\ m\ n\ \}$ of the array, where m is the number of rows and n is the number of columns.

For algebraic objects, OBJ→ returns the arguments of the top-level (least-nested) function ($arg_1\ \ \ldots\ \ arg_n$), the number of arguments

OBJ→

of the top-level function (n), and the name of the top-level function (*function*).

If the argument is a string, the object sequence defined by the string is executed.

Example: The command sequence `'∫(0,1,SIN(X),X)' OBJ→` returns:

6:	0	first argument
5:	1	second argument
4:	'SIN(X)'	third argument
3:	'X'	fourth argument
2:	4	number of arguments for $\int$
1:	∫	function name

Related Commands: ARRY→, C→R, DTAG, EQ→, LIST→, R→C, STR→, →TAG

OCT

Octal Mode Command: Selects octal base for binary integer operations. (The default base is decimal.)

Keyboard Access: (MTH) BASE OCT

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: Binary integers require the prefix #. Binary integers entered and returned in octal base automatically show the suffix □. If the current base is not octal, enter an octal number by ending it with □. It will be displayed in the current base when entered.

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Related Commands: BIN, DEC, HEX, RCWS, STWS

OFF

Off Command: Turns off the calculator.

Keyboard Access: PRG NXT FUN NXT OFF

Affected by Flags: None

Remarks: When executed from a program, that program will resume execution when the calculator is turned on. This provides a programmable "autostart."

Related Commands: CONT, HALT, KILL

OLDPRT

Old Printer Command: Modifies the remapping string in the reserved variable *PRTPAR* so that the extended character set of the HP 48 matches that of the HP 82240A Infrared Printer.

Keyboard Access: ← I/O PRINT PRTPH OLDPR

Affected by Flags: None

Remarks: The character set in the HP 82240A Infrared Printer does not match the HP 48 character set:

- 24 characters in the HP 48 character set are not available in the HP 82240A Infrared Printer. (From the table in the keyword listing for NUM, these characters are numbers 129, 130, 143-157, 159, 166, 169, 172, 174, 184, and 185.) The HP 82240A prints a ▒ in substitution.
- Many characters in the extended character table (character codes 128 through 255) do not have the same character code. For example, the ※ character has code 171 in the HP 48 and code 146 in the HP 82240A Infrared Printer.

To use the CHR command to print extended characters with an HP 82240A Infrared Printer, first execute OLDPRT. The remapping string modified by OLDPRT is the second parameter in *PRTPAR*. This string (which is empty in the default state) changes the character code of each byte to match the codes in the HP 82240A Infrared Printer character table.

OLDPRT

To cancel OLDPRT character mapping, purge the variable *PRTPAR*, or enter `na PRTPAR na 2 '' PUT`.

To print a string containing graphics data, disable OLDPRT.

Related Commands: CR, DELAY, PRLCD, PRST, PRSTC, PRVAR, PR1

OPENIO

Open I/O Port Command: Opens the serial port or the IR port using the I/O parameters in the reserved variable *IOPAR*.

Keyboard Access: ← I/O NXT SERIAL OPENIO

Affected by Flags: I/O Device (−33)

Remarks: Since all HP 48 Kermit-protocol commands automatically effect an OPENIO first, OPENIO is not normally needed, but can be used if an I/O transmission does not work. OPENIO is necessary for interaction with devices that interpret a closed port as a break.

OPENIO is also necessary for the automatic reception of data into the input buffer using non-Kermit commands. If the port is closed, incoming characters are ignored. If the port is open, incoming characters are automatically placed in the input buffer. These characters can be detected with BUFLEN, and can be read out of the input buffer using SRECV.

If the port is already open, OPENIO does not affect the data in the input buffer. However, if the port is closed, executing OPENIO clears the data in the input buffer.

For more information, refer to the reserved variable *IOPAR* in appendix D, "Reserved Variables."

Related Commands: BUFLEN, CLOSEIO, SBRK, SRECV, STIME, XMIT

OR

OR Function: Returns the logical OR of two arguments.

{}

Level 2	Level 1	→	Level 1
#n_1	#n_2	→	#n_3
"$string_1$"	"$string_2$"	→	"$string_3$"
T/F_1	T/F_2	→	0/1
T/F	'$symb$'	→	'T/F OR $symb$'
'$symb$'	T/F	→	'$symb$ OR T/F'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ OR $symb_2$'

Keyboard Access:

(MTH) BASE (NXT) LOGIC OR

(PRG) TEST (NXT) OR

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: When the arguments are binary integers or strings, OR does a bit-by-bit (base 2) logical comparison.

- An argument that is a binary integer is treated as a sequence of bits as long as the current wordsize. Each bit in the result is determined by comparing the corresponding bits (bit_1 and bit_2) in the two arguments as shown in the following table.

bit_1	bit_2	bit_1 **OR** bit_2
0	0	0
0	1	1
1	0	1
1	1	1

- An argument that is a string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must be the same length.

OR

When the arguments are real numbers or symbolics, OR simply does a true/false test. The result is 1 (true) if either or both arguments are nonzero; it is 0 (false) if both arguments are zero. This test is usually done to compare two test results.

If either or both of the arguments are algebraic objects, then the result is an algebraic of the form '$symb_1$ OR $symb_2$'. Execute →NUM (or set flag −3 before executing OR) to produce a numeric result from the algebraic result.

Related Commands: AND, NOT, XOR

ORDER

Order Variables Command: Reorders the variables in the current directory (shown in the VAR menu) to the order specified.

Level 1	→	Level 1
{ $global_1$ ... $global_n$ }	→	

Keyboard Access: ◀ (MEMORY) DIR ORDER

Affected by Flags: None

Remarks: The names that appear first in the list will be the first to appear in the VAR menu. Variables not specified in the list are placed after the reordered variables.

If the list includes the name of a large subdirectory, there may be insufficient memory to execute ORDER.

Related Commands: VARS

OVER

Over Command: Returns a copy to stack level 1 of the object in level 2.

Level 2	Level 1	→	Level 3	Level 2	Level 1
obj_1	obj_2	→	obj_1	obj_2	obj_1

Keyboard Access: ◀ STACK OVER

Affected by Flags: None

Related Commands: PICK, ROLL, ROLLD, ROT, SWAP

PARAMETRIC

Parametric Plot Type Command: Sets the plot type to PARAMETRIC.

Keyboard Access: ◀ PLOT PTYPE PARA

Affected by Flags: Simultaneous Plotting (−28), Curve Filling (−28)

Remarks: When the plot type is PARAMETRIC, the DRAW command plots the current equation as a complex-valued function of one real variable. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

$$\{\ (x_{min}, y_{min})\ (x_{max}, y_{max})\ indep\ res\ axes\ ptype\ depend\ \}$$

For plot type PARAMETRIC, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (−6.5,−3.1).

- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).

- *indep* is a list containing a name that specifies the independent variable, and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). Note that the default value is *X*. If *X* is not modified and included in a list with a plotting range, the values in (x_{min}, y_{min}) and (x_{max}, y_{max}) are used as the plotting range, which generally leads to meaningless results.

- *res* is a real number specifying the interval, in user-unit coordinates, between values of the independent variable. The default value is 0, which specifies an interval equal to 1/130 of the difference between the maximum and minimum values in *indep* (the plotting range).

- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).

- *ptype* is a command name specifying the plot type. Executing the command PARAMETRIC places the name PARAMETRIC in *PPAR*.

- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The contents of *EQ* must be an expression or program; it cannot be an equation. It is evaluated for each value of the independent variable. The results, which must be complex numbers, give the coordinates of the points to be plotted. Lines are drawn between plotted points unless flag −31 is set.

If flag −28 is set, all equations are plotted simultaneously.

See chapter 23 of the *HP 48 User's Guide* for an example using the PARAMETRIC plot type.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

PARITY

Parity Command: Sets the parity value in the reserved variable *IOPAR*.

{}

Level 1	→	Level 1
n_{parity}	→	

Keyboard Access: ◀ I/O IOPAR PARIT

Affected by Flags: None

Remarks: Legal values are shown below. A negative value means the HP 48 does not check parity on bytes received during Kermit transfers or with SRECV. Parity is still used during data transmission, however.

n-Value	Meaning
0	no parity (the default value)
1	odd parity
2	even parity
3	mark
4	space

For more information, refer to the reserved variable *IOPAR* (*I/O parameters*) in appendix D, "Reserved Variables."

Related Commands: BAUD, CKSM, TRANSIO

PARSURFACE

PARSURFACE Plot Type Command: Sets plot type to PARSURFACE.

Keyboard Access: ◀ PLOT NXT 3D PTYPE PARSU

Affected by Flags: None

Remarks: When plot type is set to PARSURFACE, the DRAW command plots an image graph of a 3-vector-valued function of two variables. PARSURFACE requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR is made up of the following elements:

{ x_{left} x_{right} y_{near} y_{far} z_{low} z_{high} x_{min} x_{max} y_{min} y_{max} x_{eye} y_{eye} z_{eye} x_{step} y_{step} }

For plot type PARSURFACE, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are real numbers the specify the input region's width. The default value is (-1,1).
- y_{min} and y_{max} are real numbers that specify the input region's depth. The default value is (-1,1).
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

For plot type PARSURFACE, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is not used.
- (x_{max}, y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is not used.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command PARSURFACE places the name PARSURFACE in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

PATH

Current Path Command: Returns a list specifying the path to the current directory.

Level 1 →	**Level 1**
→	{ HOME *directory-name*$_1$ ... *directory-name*$_n$ }

Keyboard Access: ◀ MEMORY DIR PATH

Affected by Flags: None

Remarks: The first directory is always *HOME*, and the last directory is always the current directory.

If a program needs to switch to a specific directory, it can do so by evaluating a directory list, such as one created earlier by PATH.

PATH

Related Commands: CRDIR, HOME, PGDIR, UPDIR

PCOEF

Monic Polynomial Coefficients Command: Returns the coefficients of a monic polynomial (a polynomial with a leading coefficient of 1) having specific roots.

{ }

Level 1	→	Level 1
[*array*]$_{roots}$	→	[*array*]$_{coefficients}$

Keyboard Access: ◀ SOLVE POLY PCOEF

Affected by Flags: None

Remarks: The argument must be a real or complex array of length n containing the polynomial's roots. The result is a real or complex vector of length $n+1$ containing the coefficients listed from highest order to lowest, with a leading coefficient of 1.

Example: Find the polynomial that has the roots 2, −3, 4, −5:

`[ 2 -3 4 -5 ] PCOEF` returns `[ 1 2 -25 -26 120 ]`, representing the polynomial $x^4 + 2x^3 - 25x^2 - 26x + 120$.

Related Commands: PEVAL, PROOT

PCONTOUR

PCONTOUR Plot Type Command: Sets the plot type to PCONTOUR.

Keyboard Access: ◀ PLOT NXT 3D PTYPE PCON

Affected by Flags: None

Remarks: When plot type is set PCONTOUR, the DRAW command plots a contour-map view of a scalar function of two variables. PCONTOUR requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR is made up of the following elements:

{ x_{left} x_{right} y_{near} y_{far} z_{low} z_{high} x_{min} x_{max} y_{min} y_{max} x_{eye} y_{eye} z_{eye} x_{step} y_{step} }

For plot type PCONTOUR, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

For plot type PCONTOUR, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is not used.
- (x_{max}, y_{max}) is not used.

- *indep* is a name specifying the independent variable. The default value of *indep* is X.
- *res* is not used.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command PCONTOUR places the name PCONTOUR in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is Y.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

PCOV

Population Covariance Command: Returns the population covariance of the independent and dependent data columns in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	$x_{pcovariance}$

Keyboard Access: ↰ STAT FIT NXT PCOV

Affected by Flags: None

Remarks: The columns are specified by the first two elements in reserved variable ΣPAR, set by XCOL and YCOL respectively. If ΣPAR does not exist, PCOV creates it and sets the elements to their default values (1 and 2).

The population covariance is calculated with the following formula:

$$\frac{1}{n}\sum_{k=1}^{n}(x_{kn_1}-\overline{x_{n_1}})(x_{kn_2}-\overline{x_{n_2}})$$

where x_{kn_1} is the kth coordinate value in column n_1, x_{kn_2} is the kth coordinate value in the column n_2, $\overline{x_{n_1}}$ is the mean of the data in column n_1, $\overline{x_{n_2}}$ is the mean of the data in column n_2, and n is the number of data points.

Related Commands: COLΣ, CORR, COV, PREDX, PREDY, XCOL, YCOL

PDIM

PICT Dimension Command: Replaces *PICT* with a blank *PICT* of the specified dimensions.

{ }

Level 2	Level 1	→	Level 1
(x_{min}, y_{min})	(x_{max}, y_{max})	→	
$\#n_{width}$	$\#m_{height}$	→	

Keyboard Access: PRG PICT PDIM

Affected by Flags: None

Remarks: If the arguments are complex numbers, PDIM changes the size of *PICT* and makes the arguments the new values of (x_{min}, y_{min}) and (x_{max}, y_{max}) in the reserved variable *PPAR*. Thus, the scale of a subsequent plot is not changed. If the arguments are binary integers, *PPAR* remains unchanged, so the scale of a subsequent plot *is* changed.

PICT cannot be smaller than 131 pixels wide × 64 pixels high, nor wider than 2048 pixels (height is unlimited).

Related Commands: PMAX, PMIN

PERM

Permutations Function: Returns the number of possible permutations of n items taken m at a time.

{ }

Level 2	Level 1	→	Level 1
n	m	→	$P_{n,m}$
'$symb_n$'	m	→	'PERM($symb_{n,m}$)'
n	'$symb_m$'	→	'PERM(n, $symb_m$)'
'$symb_n$'	'$symb_m$'	→	'PERM($symb_n$, $symb_m$)'

Keyboard Access: MTH NXT PROB PERM

Affected by Flags: Numerical Results (−3)

Remarks: The formula used to calculate $P_{n,m}$ is this:

$$P_{n,m} = \frac{n!}{(n-m)!}$$

The arguments n and m must each be less than 10^{12}.

Related Commands: COMB, !

PEVAL

Polynomial Evaluation Command: Evaluates an n-degree polynomial at x.

{ }

Level 2	Level 1	→	Level 1
[*array*]$_{coefficients}$	x	→	$p(x)$

Keyboard Access: ← SOLVE POLY PEVAL

Affected by Flags: None

Remarks: The arguments must be an array of length $n+1$ containing the polynomial's coefficients listed from highest order to lowest, and the value x at which the polynomial is to be evaluated.

Example: Evaluate the polynomial $x^4 + 2x^3 - 25x^2 - 26x + 120$ at $x = 8$:

`[ 1 2 -25 -26 120 ] 8` returns `3432`.

Related Commands: PCOEF, PROOT

PGDIR

Purge Directory Command: Purges the named directory (whether empty or not).

{ }

Level 1	→	Level 1
'global'	→	

Keyboard Access: ↰ MEMORY DIR PGDIR

Affected by Flags: None

Related Commands: CLVAR, CRDIR, HOME, PATH, PURGE, UPDIR

PICK

Pick Object Command: Copies the contents of a specified level to level 1.

Level n+1..	Level 2	Level 1	→	Level n+1..	Level 2	Level 1
obj_n ..	obj_1	n	→	obj_n ..	obj_1	obj_n

Keyboard Access: ◀ STACK PICK

Affected by Flags: None

Related Commands: DUP, DUPN, DUP2, OVER, ROLL, ROLLD, ROT, SWAP

PICT

PICT Command: Puts the name PICT on the stack.

Level 1	→	Level 1
	→	*PICT*

Keyboard Access: PRG PICT PICT

Affected by Flags: None

Remarks: *PICT* is the name of a storage location in calculator memory containing the current graphics object. The command PICT enables access to the contents of that memory location as if it were a variable. Note, however, that *PICT* is *not* a variable as defined in the HP 48: its name cannot be quoted, and only graphics objects may be "stored" in it.

If a graphics object smaller than 131 wide × 64 pixels high is stored in *PICT*, it is enlarged to 131 × 64. A graphics object of unlimited pixel height and up to 2048 pixels wide can be stored in *PICT*.

Examples: PICT RCL returns the current graphics object to the stack.

GRAPHIC 131 × 64 PICT STO stores a graphics object in *PICT*, making it the current graphics object.

Related Commands: GOR, GXOR, NEG, PICTURE, PVIEW, RCL, REPL, SIZE, STO, SUB

PICTURE

Picture Environment Command: Selects the Picture environment (selects the graphics display and activates the graphics cursor and Picture menu).

Keyboard Access: ◀ PICTURE

Affected by Flags: None

Remarks: When executed from a program, PICTURE suspends program execution until CANCEL is pressed.

Example: This program:

```
« "Press CANCEL to return■to stack" 1 DISP
3 WAIT PICTURE »
```

displays a message for 3 seconds, then selects the Picture environment. (The ■ character in the program indicates a linefeed.)

Related Commands: PICTURE, PVIEW, TEXT

PINIT

Port Initialize Command: Initializes all currently active ports. Does not affect data already stored in a port.

Keyboard Access: ◀ LIBRARY NXT PINIT

Affected by Flags: None

Related Commands: None

Remarks: PINIT is particularly useful when using a plug-in card that can hold multiple ports. It stores and then purges an object in each port (128K partition) that can be accessed at the time the command is executed. This has the effect of initializing each port without disturbing any previously-stored data.

PIXOFF

Pixel Off Command: Turns off the pixel at the specified coordinate in *PICT*.

Level 1	→	Level 1
(*x*, *y*)	→	
{ #*n* #*m* }	→	

Keyboard Access: PRG PICT NXT PIXOF

Affected by Flags: None

Related Commands: PIXON, PIX?

PIXON

Pixel On Command: Turns on the pixel at the specified coordinate in *PICT*.

Level 1	→	Level 1
(*x*, *y*)	→	
{ #*n* #*m* }	→	

Keyboard Access: PRG PICT NXT PIXON

Affected by Flags: None

Related Commands: PIXOFF, PIX?

PIX?

Pixel On? Command: Tests whether the specified pixel in *PICT* is on; returns 1 (true) if the pixel is on, and 0 (false) if the pixel is off.

Level 1	→	Level 1
(*x*, *y*)	→	0/1
{ #*n* #*m* }	→	0/1

Keyboard Access: PRG PICT NXT PIX?

Affected by Flags: None

Related Commands: PIXON, PIXOFF

PKT

Packet Command: Used to send command "packets" (and receive requested data) to a Kermit server.

{ }

Level 2	Level 1	→	Level 1
"*data*"	"*type*"	→	"*response*"

Keyboard Access: ← I/O SRVR PKT

Affected by Flags: I/O Device (−33), I/O Messages (−39)

The I/O Data Format flag (−35) can be significant if the server sends back more than one packet.

Remarks: To send HP 48 objects, use SEND.

PKT allows additional commands to be sent to a Kermit server. For more information, refer to *Using MS-DOS Kermit* by Christine M. Gianone, Digital Press, 1990; or *KERMIT, A File Transfer Protocol* by Frank da Cruz, Digital Press, 1987, especially chapter 11, "The Client/Server Model."

The packet data, packet type, and the response to the packet transmission are all in string form. PKT first does an I (*initialization*) packet exchange with the Kermit server, then sends the server a packet constructed from the data and packet-type arguments supplied to PKT. The response to PKT will be either an acknowledging string (possibly blank) or an error packet (see KERRM).

For the *type* argument, only the first letter is significant.

Examples: A PKT command to send a *g*eneric *d*irectory request is "D" "G" PKT.

To send a *host command* packet, use a command from the server's operating system for the *data* string and "C" for the *type* string. For example, "'ABC' PURGE" "C" PKT on a local HP 48 would instruct an HP 48 server to purge variable *ABC*.

Related Commands: CLOSEIO, KERRM, SERVER

PMAX

PICT Maximum Command: Specifies (x, y) as the coordinates at the upper right corner of the display.

{ }

Level 1	→	Level 1
(x, y)	→	

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: The complex number (x, y) is stored as the second element in the reserved variable $PPAR$.

Related Commands: PDIM, PMIN, XRNG, YRNG

PMIN

PICT Minimum Command: Specifies (x, y) as the coordinates at the lower left corner of the display.

{ }

Level 1	→	Level 1
(x,y)	→	

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: The complex number (x, y) is stored as the first element in the reserved variable $PPAR$.

Related Commands: PDIM, PMAX, XRNG, YRNG

POLAR

Polar Plot Type Command: Sets the plot type to POLAR.

Keyboard Access: ◀ PLOT PTYPE POLAR

Affected by Flags: Simultaneous Plotting (−28), Curve Filling (−31)

Remarks: When the plot type is POLAR, the DRAW command plots the current equation in polar coordinates, where the independent variable is the polar angle and the dependent variable is the radius. The current equation is specified in the reserved variable *EQ*.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

For plot type POLAR, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (−6.5,−3.1).
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value of *indep* is *X*.
- *res* is a real number specifying the interval, in user-unit coordinates, between values of the independent variable. The default value is 0, which specifies an interval of 2 degrees, 2 grads, or $\pi/90$ radians.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command POLAR places the name POLAR in *ptype*.

- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The current equation is plotted as a function of the variable specified in *indep*. The minimum and maximum values of the independent variable (the plotting range) can be specifed in *indep*; otherwise, the default minimum value is 0 and the default maximum value corresponds to one full circle in the current angle mode (360 degrees, 400 grads, or 2π radians). Lines are drawn between plotted points unless flag −31 is set.

If flag −28 is set, all equations are plotted simultaneously.

If *EQ* contains an expression or program, the expression or program is evaluated in Numerical Results mode for each value of the independent variable to give the values of the dependent variable. If *EQ* contains an equation, the plotting action depends on the form of the equation.

Form of Current Equation	Plotting Action
'*expr*=*expr*'	Each expression is plotted separately. The intersection of the two graphs shows where the expressions are equal.
'*name*=*expr*'	Only the expression is plotted.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

POS

Position Command: Returns the position of a substring within a string or the position of an object within a list.

Level 2	Level 1	→	Level 1
"string"	*"substring"*	→	*n*
{ list }	*obj*	→	*n*

Keyboard Access:

◀ CHARS POS

PRG LIST ELEM POS

Affected by Flags: None

Remarks: If there is no match for *obj* or *substring*, POS returns zero.

Related Commands: CHR, NUM, REPL, SIZE, SUB

PREDV

Predicted y-Value Command: Returns the predicted dependent-variable value $y_{\mathsf{dependent}}$, based on the independent-variable value $x_{\mathsf{independent}}$, the currently selected statistical model, and the current regression coefficients in the reserved variable ΣPAR.

{ }

Level 1	→	Level 1
$x_{\mathsf{independent}}$	→	$y_{\mathsf{dependent}}$

Keyboard Access: None. Must be typed in.

Remarks: Provided for compatibility with the HP 28. PREDV is the same as PREDY. See PREDY.

PREDX

Predicted x-Value Command: Returns the predicted independent-variable value $x_{independent}$, based on the dependent-variable value $y_{dependent}$, the currently selected statistical model, and the current regression coefficients in the reserved variable ΣPAR.

{ }

Level 1	→	Level 1
$y_{dependent}$	→	$x_{independent}$

Keyboard Access: ↩ STAT FIT PREDX

Affected by Flags: None

Remarks: The value is predicted using the regression coefficients most recently computed with LR and stored in the reserved variable ΣPAR. For the linear statistical model, the equation used is this:

$$y_{dependent} = (mx_{independent}) + b$$

where m is the slope (the third element in ΣPAR) and b is the intercept (the fourth element in ΣPAR).

For the other statistical models, the equations used by PREDX are listed in the LR entry.

If PREDX is executed without having previously generated regression coefficients in ΣPAR, a default value of zero is used for both regression coefficients, and an error results.

Example: Given five columns of data in ΣDAT, the command sequence:

```
2 XCOL 5 YCOL LOGFIT LR 23 PREDX
```

sets column 2 as the independent variable column, sets column 5 as the dependent variable column, and sets the logarithmic statistical

model. It then executes LR, generating intercept and slope regression coefficients, and storing them in ΣPAR. Then, given a dependent value of 23, it returns a predicted independent value based on the regression coefficients and the statistical model.

Related Commands: COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, LR, PREDY, PWRFIT, XCOL, YCOL

PREDY

Predicted y-Value Command: Returns the predicted dependent-variable value $y_{\mathsf{dependent}}$, based on the independent-variable value $x_{\mathsf{independent}}$, the currently selected statistical model, and the current regression coefficients in the reserved variable ΣPAR.

{ }

Level 1	→	Level 1
$x_{\mathsf{independent}}$	→	$y_{\mathsf{dependent}}$

Keyboard Access: ↩ STAT FIT PREDY

Affected by Flags: None

Remarks: The value is predicted using the regression coefficients most recently computed with LR and stored in the reserved variable ΣPAR. For the linear statistical model, the equation used is this:

$$y_{dependent} = (mx_{independent}) + b$$

where m is the slope (the third element in ΣPAR) and b is the intercept (the fourth element in ΣPAR).

For the other statistical models, the equations used by PREDY are listed in the LR entry.

If PREDY is executed without having previously generated regression coefficients in ΣPAR, a default value of zero is used for both regression coefficients—in this case PREDY will return 0 for statistical models LINFIT and LOGFIT, and error for statistical models EXPFIT and PWRFIT.

Example: Given four columns of data in ΣDAT, the command sequence:

```
2 XCOL 4 YCOL PWRFIT LR 11 PREDY
```

sets column 2 as the independent variable column, sets column 4 as the dependent variable column, and sets the power statistical model. It then executes LR, generating intercept and slope regression coefficients, and storing them in ΣPAR. Then, given an independent value of 11, it returns a predicted dependent value based on the regression coefficients and the statistical model.

Related Commands: COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, LR, PREDX, PWRFIT, XCOL, YCOL

PRLCD

Print LCD Command: Prints a pixel-by-pixel image of the current display (excluding the annunciators).

Keyboard Access: ↰ I/O PRINT PRLCD

Affected by Flags: Printing Device (−34), I/O Device (−33), Linefeed (−38)

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

Flag −38 must be clear.

Remarks: The width of the printed image of characters in the display is narrower using PRLCD than using a print command such as PR1. The difference results from the spacing between characters. On the display there is a single blank column between characters, and PRLCD prints this spacing. Print commands such as PR1 print two blank columns between adjacent characters.

Example: The command sequence ERASE DRAW PRLCD clears $PICT$, plots the current equation, then prints the graphics display.

Related Commands: CR, DELAY, OLDPRT, PRST, PRSTC, PRVAR, PR1

PROMPT

Prompt Command: Displays the contents of "*prompt*" in the status area, and halts program execution.

{ }

Level 1	→	Level 1
"*prompt*"	→	

Keyboard Access: PRG NXT I/O NXT PROMPT

Affected by Flags: None

Remarks: PROMPT is equivalent to 1 DISP 1 FREEZE HALT.

Related Commands: CONT, DISP, FREEZE, HALT, INFORM, INPUT, MSGBOX

PROOT

Polynomial Roots Command: Returns all roots of an n-degree polynomial having real or complex coefficients.

{ }

Level 1	→	Level 1
[*array*]$_{coefficients}$	→	[*array*]$_{roots}$

Keyboard Access: ↶ SOLVE POLY PROOT

Affected by Flags: Infinite Result Exception (−22)

Remarks: For an n^{th}-order polynomial, the argument must be a real or complex array of length $n+1$ containing the coefficients listed from highest order to lowest. The result is a real or complex vector of length n containing the computed roots.

PROOT interprets leading coefficients of zero in a limiting sense. As a leading coefficient approaches zero, a root of the polynomial approaches infinity: therefore, if flag −22 is clear (the default), PROOT reports an Infinite Result error if a leading coefficient is zero. If flag −22 is set, PROOT returns a root of (MAXREAL,0) for each leading zero in an array containing real coefficients, and a root of (MAXREAL,MAXREAL) for each leading zero in an array containing complex coefficients.

Example: Find the roots of the polynomial $x^4+2x^3-25x^2-26x+120$:

`[ 1 2 -25 -26 120 ]` `PROOT` returns `[ 2 -3 4 -5 ]`.

Related Commands: PCOEF, PEVAL

PRST

Print Stack Command: Prints all objects in the stack, starting with the object in the highest level.

Keyboard Access: ◀ I/O PRINT PRST

Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33), Linefeed (−38)

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

When flag −38 is set, linefeeds are *not* added at the end of each print line. Generally, flag −38 should be clear for execution of PRST. PRST leaves the stack unchanged.

Remarks: Objects are printed in multiline printer format. See the PR1 entry for a description of multiline printer format.

Related Commands: CR, DELAY, OLDPRT, PRLCD, PRSTC, PRVAR, PR1

PRSTC

Print Stack (Compact) Command: Prints in compact form all objects in the stack, starting with the object in the highest level.

Keyboard Access: ← I/O PRINT PRSTC

Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33), Linefeed (−38)

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

When flag −38 is set, linefeeds are *not* added at the end of each print line. Generally, flag −38 should be clear for execution of PRSTC.

Remarks: Compact printer format is the same as compact display format. Multiline objects are truncated and appear on one line only. PRSTC leaves the stack unchanged.

Related Commands: CR, DELAY, OLDPRT, PRLCD, PRST, PRVAR, PR1

PRVAR

Print Variable Command: Searches the current directory path or port for the specified variables and prints the name and contents of each variable.

Level 1	→	Level 1
'*name*'	→	
{ $name_1$ $name_2$ ... }	→	
:n_{port} :'*global*'	→	

Keyboard Access: ← I/O PRINT PRVAR

Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33), Linefeed (−38)

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

When flag −38 is set, linefeeds are *not* added at the end of each print line. Generally, flag −38 should be clear for execution of PRVAR.

Remarks: Objects are printed in multiline printer format. See the PR1 entry for a description of multiline printer format.

Related Commands: CR, DELAY, OLDPRT, PR1, PRLCD, PRST, PRSTC

PR1

Print Level 1 Command: Prints an object in multiline printer format.

Keyboard Access: ← I/O PR1

Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33), Linefeed (−38)

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

Remarks: All objects except strings are printed with their identifying delimiters. Strings are printed without the leading and trailing " delimiters. PR1 leaves the stack unchanged.

Multiline printer format is similar to multiline display format, with the following exceptions:

- Strings and names that are more than 24 characters long are continued on the next printer line.
- The real and imaginary parts of complex numbers are printed on separate lines if they don't fit on the same line.
- Arrays are printed with a numbered heading for each row and with a column number before each element. For example, the 2 × 3 array

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

would be printed as follows:

```
Array { 2 3 } ── Array dimensions
Row number ── Row 1
Column number { 1] 1
                2] 2
                3] 3
Row 2
1] 4
2] 5
3] 6
```

Related Commands: CR, DELAY, OLDPRT, PRLCD, PRST, PRSTC, PRVAR

PSDEV

Population Standard Deviation Command: Calculates the population standard deviation of each of the m columns of coordinate values in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{psdev}
	→	$[\ x_{psdev1}\ x_{psdev2}\ \cdots\ x_{psdevm}\ ]$

Keyboard Access: ◀ STAT 1'HR NXT PSDE'

Affected by Flags: None

Remarks: PSDEV returns a vector of m real numbers, or a single real number if $m = 1$. The population standard deviation is computed using this formula:

$$\sqrt{\frac{1}{n}\sum_{k=1}^{n}(x_k - \overline{x})^2}$$

where x_k is the kth coordinate value in a column, $\overline{x}$ is the mean of the data in this column, and n is the number of data points.

Related Commands: MEAN, PCOV, PVAR, SDEV, TOT, VAR

PURGE

Purge Command: Purges the named variables or empty subdirectories from the current directory.

Level 1	→	Level 1
'*global*'	→	
{ $global_1$ ... $global_n$ }	→	
PICT	→	
:n_{port} :$name_{backup}$	→	
:n_{port} :$n_{library}$	→	

Keyboard Access: ◀ PURGE

Affected by Flags: None

Remarks: PURGE executed in a program does not save its argument for recovery by LASTARG.

To empty a named directory before purging it, use PGDIR.

To help prepare a list of variables for purging, use VARS.

Purging *PICT* replaces the current graphics object with a 0×0 graphics object.

If a list of objects (with global names, backup objects, library objects, or *PICT*) for purging contains an invalid object, then the objects

preceding the invalid object are purged, and the error `Bad Argument Type` occurs.

To purge a library or backup object, tag the library number or backup name with the appropriate port number ($:n_{port}$), which must be in the range from 0 to 33. (A library can be purged from RAM only.) For a backup object, the port number can be replaced with the wildcard character &, in which case the HP 48 will search ports 33 through 0, and then main memory for the named backup object.

Library objects in RAM can be purged, while those in ROM (application cards and write-protected RAM cards) cannot. A library object must be detached before it can be purged from the *HOME* directory.

Neither a library object nor a backup object can be purged if it is currently "referenced" internally by stack pointers (such as an object on the stack, in a local variable, on the LAST stack, or on an internal return stack). This produces the error `Object in Use`. To avoid these restrictions, use NEWOB before purging. (See NEWOB.)

Related Commands: CLEAR, CLUSR, CLVAR, NEWOB, PGDIR

PUT

Put Element Command: In the level 3 array or list, PUT replaces with z_{put} or obj_{put} the object whose position is specified in level 2; if the array or list is unnamed, returns the new array or list.

Level 3	Level 2	Level 1	→	Level 1
[[*matrix*]]$_1$	$n_{position}$	z_{put}	→	[[*matrix*]]$_2$
[[*matrix*]]$_1$	{ n_{row} m_{col} }	z_{put}	→	[[*matrix*]]$_2$
'*name*$_{matrix}$'	$n_{position}$	z_{put}	→	
'*name*$_{matrix}$'	{ n_{row} m_{col} }	z_{put}	→	
[*vector*]$_1$	$n_{position}$	z_{put}	→	[*vector*]$_2$
[*vector*]$_1$	{ $n_{position}$ }	z_{put}	→	[*vector*]$_2$
'*name*$_{vector}$'	$n_{position}$	z_{put}	→	
'*name*$_{vector}$'	{ $n_{position}$ }	z_{put}	→	
{ *list* }$_1$	$n_{position}$	obj_{put}	→	{ *list* }$_2$
{ *list* }$_1$	{ $n_{position}$ }	obj_{put}	→	{ *list* }$_2$
'*name*$_{list}$'	$n_{position}$	obj_{put}	→	
'*name*$_{list}$'	{ $n_{position}$ }	obj_{put}	→	

Keyboard Access: PRG LIST ELEM PUT

Affected by Flags: None

Remarks: For matrices, $n_{position}$ counts in row order.

If the argument in level 3 is a name, PUT alters the named array or list and returns nothing to the stack.

Examples: This command sequence:
`[[ 2 3 4 ][ 4 1 2 ]] ( 1 3 ) 96 PUT` returns
`[[ 2 3 96 ][ 4 1 2 ]]`.

The command sequence `[[ 2 3 4 ][ 4 1 2 ]] 5 96 PUT` returns
`[[ 2 3 4 ][ 4 96 2 ]]`.

The command sequence `( A B C D E ) ( 3 ) 'Z' PUT` returns
`( A B Z D E )`.

Related Commands: GET, GETI, PUTI

PUTI

Put and Increment Index Command: In the level 3 array or list, replaces with z_{put} or obj_{put} the object whose position is specified in level 2, returning the new array or list *and* the next position in that array or list.

Level 3	Level 2	Level 1	→	Level 2	Level 1
[[*matrix*]]$_1$	n_{pos1}	z_{put}	→	[[*matrix*]]$_2$	n_{pos2}
[[*matrix*]]$_1$	{ n_r m_c }$_1$	z_{put}	→	[[*matrix*]]$_2$	{ n_r m_c }$_2$
'*name*$_{matrix}$'	n_{pos1}	z_{put}	→	'*name*$_{matrix}$'	n_{pos2}
'*name*$_{matrix}$'	{ n_r m_c }$_1$	z_{put}	→	'*name*$_{matrix}$'	{ n_r m_c }$_2$
[*vector*]$_1$	n_{pos1}	z_{put}	→	[*vector*]$_2$	n_{pos2}
[*vector*]$_1$	{ n_{pos1} }	z_{put}	→	[*vector*]$_2$	{ n_{pos2} }
'*name*$_{vector}$'	n_{pos}	z_{put}	→	'*name*$_{vector}$'	n_{pos2}
'*name*$_{vector}$'	{ n_{pos1} }	z_{put}	→	'*name*$_{vector}$'	{ n_{pos2} }
{ *list* }$_1$	n_{pos1}	obj_{put}	→	{ *list* }$_2$	n_{pos2}
{ *list* }$_1$	{ n_{pos1} }	obj_{put}	→	{ *list* }$_2$	{ n_{pos2} }
'*name*$_{list}$'	n_{pos1}	obj_{put}	→	'*name*$_{list}$'	n_{pos2}
'*name*$_{list}$'	{ n_{pos1} }	obj_{put}	→	'*name*$_{list}$'	{ n_{pos2} }

Keyboard Access: (PRG) LIST ELEM PUTI

Affected by Flags: Index Wrap Indicator (−64)

The Index Wrap Indicator flag is cleared on each execution of PUTI *until* the position (index) wraps to the first position in the array or list, at which point the flag is set. The next execution of PUTI again clears the flag.

Remarks: For matrices, the position is incremented in *row* order.

Unlike PUT, PUTI returns a named array or list (to level 2). This enables a subsequent execution of PUTI at the next position of a named array or list.

Example: The following program uses PUTI and flag −64 to replace A, B, and C in the list with X.

```
« { A B C } DO 'X' PUTI UNTIL -64 FS? END »
```

Related Commands: GET, GETI, PUT

PVAR

Population Variance Command: Calculates the population variance of the coordinate values in each of the m columns in the current statistics matrix (ΣDAT).

Level 1	→	Level 1
	→	$x_{pvariance}$
	→	$[\ x_{pvariance1} \ \cdots \ x_{pvariancem}\ ]$

Keyboard Access: ◀ STAT 1'VAR NXT P'VAR

Affected by Flags: None

Remarks: The population variance (equal to the square of the population standard deviation) is returned as a vector of m real numbers, or as a single real number if $m = 1$. The population variances are computed using this formula:

$$\frac{1}{n}\sum_{k=1}^{n}(x_k - \overline{x})^2$$

where x_k is the kth coordinate value in a column, $\overline{x}$ is the mean of the data in this column, and n is the number of data points.

Related Commands: MEAN, PCOV, PSDEV, SDEV, VAR

PVARS

Port-Variables Command: Returns a list of the backup objects (:n_{port}:*name*) and the library objects (:n_{port}:$n_{library}$) in the specified port. Also returns the available memory size (if RAM) or the memory type.

{ }

Level 1	→	Level 2	Level 1
n_{port}	→	{ :n_{port} :$name_{backup}$ ... }	*memory*
n_{port}	→	{ :n_{port} :$n_{library}$ ... }	*memory*

Keyboard Access: ↰ LIBRARY PVARS

Affected by Flags: None

Remarks: The port number, n_{port}, must be in the range from 0 to 33.

- If $n_{port} = 0$, then *memory* is bytes of available main RAM.
- If the port contains independent RAM, then *memory* is bytes of available RAM in that port.
- If the port contains merged RAM, then *memory* is "SYSRAM".
- If the port contains ROM, then *memory* is "ROM".
- If the port is empty, then the message Port Not Available appears.

Related Commands: PVARS, VARS

PVIEW

PICT View Command: Displays *PICT* with the specified coordinate at the upper left corner of the graphics display.

Level 1	→	Level 1
(*x*, *y*)	→	
{ #*n* #*m* }	→	
{ }	→	

Keyboard Access:

PRG NXT OUT PVIEW

PRG PICT NXT PVIEW

Affected by Flags: None

Remarks: *PICT* must fill the entire display on execution of PVIEW. Thus, if a position other than the upper left corner of *PICT* is specified, *PICT* must be large enough to fill a rectangle that extends 131 pixels to the right and 64 pixels down.

If PVIEW is executed from a program with a coordinate argument (versus an empty list), the graphics display persists only until the keyboard is ready for input (for example, until the end of program execution). However, the FREEZE command freezes the display until a key is pressed.

If PVIEW is executed with an *empty* list argument, *PICT* is centered in the graphics display with scrolling mode activated. In this case, the graphics display persists until CANCEL is pressed.

PVIEW does *not* activate the graphics cursor or the Picture menu. To activate the graphics cursor and Picture menu, execute PICTURE.

Example: The program

```
« { # 0d # 0d } PVIEW  7 FREEZE »
```

displays *PICT* in the graphics display with coordinates { # 0d # 0d } in the upper left corner of the display, then freezes the full display until a key is pressed.

PVIEW

Related Commands: FREEZE, PICTURE, TEXT

PWRFIT

Power Curve Fit Command: Stores PWRFIT as the fifth parameter in the reserved variable ΣPAR, indicating that subsequent executions of LR are to use the power curve fitting model.

Keyboard Access: ◀ STAT ΣPAR MODL PWRFI

Affected by Flags: None

Remarks: LINFIT is the default specification in ΣPAR. For a description of ΣPAR, see appendix D, "Reserved Variables."

Related Commands: BESTFIT, EXPFIT, LINFIT, LOGFIT, LR

PX→C

Pixel to Complex Command: Converts the specified pixel coordinates to user-unit coordinates.

Level 1	→	Level 1
{ #*n* #*m* }	→	(*x*, *y*)

Keyboard Access: PRG PICT NXT PX→C

Affected by Flags: None

Remarks: The user-unit coordinates are derived from the $\{ x_{min}, y_{min} \}$ and $\{ x_{max}, y_{max} \}$ parameters in the reserved variable $PPAR$. The coordinates correspond to the geometrical center of the pixel.

Related Commands: C→PX

→Q

To Quotient Command: Returns a rational form of the argument.

{ }

Level 1	→	Level 1
x	→	'*a*/*b*'
(*x*,*y*)	→	'*a*/*b*+*c*/*d**i'
'$symb_1$'	→	'$symb_2$'

Keyboard Access: ◀ SYMBOLIC NXT +□

Affected by Flags: Number Display (−45 to −50)

Remarks: The rational result is a "best guess", since there might be more than one rational expression consistent with the argument. →Q finds a quotient of integers that agrees with the argument to within the number of decimal places specified by the display format mode.

→Q also acts on numbers that are part of algebraic expressions or equations.

Example: '√+2.5' →Q returns '√+5/2'.

Related Commands: →Qπ, /

→Qπ

To Quotient Times π Command: Returns a rational form of the argument, *or* a rational form of the argument with π factored out, whichever yields the smaller denominator.

→Qπ

{ }

Level 1	→	Level 1
x	→	'*a*/*b**π'
x	→	'*a*/*b*'
'*symb*$_1$'	→	'*symb*$_2$'
(*x*,*y*)	→	'a/b*π+c/d*π*i'
(*x*,*y*)	→	'a/b+c/d*i'

Keyboard Access: ↰ SYMBOLIC NXT → ℓ! π

Affected by Flags: Number Format (−45 to −50)

Remarks: →Qπ computes two quotients (rational expressions) and compares them: the quotient of the argument, and the quotient of the argument divided by π. It returns the fraction with the smaller denominator; if the argument was divided by π, then π is a factor in the result.

The rational result is a "best guess," since there might be more than one rational expression consistent with the argument. →Qπ finds a quotient of integers that agrees with the argument to the number of decimal places specified by the display format mode.

→Qπ also acts on numbers that are part of algebraic expressions or equations.

For a complex argument, the real or imaginary part (or both) can have π as a factor.

Example: In Fix mode to four decimal places, `6.2832 →Qπ` returns `'2*π'`. In Standard mode, however, `6.2832 →Qπ` returns `3927/625`.

Related Commands: →Q, /, π

QR

QR Factorization of a Matrix Command: Returns the QR factorization of an $n \times m$ matrix.

{ }

Level 1	→	Level 3	Level 2	Level 1
[[*matrix*]]$_A$	→	[[*matrix*]]$_Q$	[[*matrix*]]$_R$	[[*matrix*]]$_P$

Keyboard Access: (MTH) MATR FACTR QR

Affected by Flags: None

Remarks: QR factors $m \times n$ matrix A into three matrices:

- Q is an $m \times m$ orthogonal matrix.
- R is an $m \times n$ upper trapezoidal matrix.
- P is a $n \times n$ permutation matrix.

Where $A \times P = Q \times R$.

Related Commands: LQ, LSQ

QUAD

Solve Quadratic Equation Command: Solves an algebraic object '$symb_1$' for the variable *global*, and returns an expression '$symb_2$' representing the solution.

{ }

Level 2	Level 1	→	Level 1
'$symb_1$'	'*global*'	→	'$symb_2$'

Keyboard Access: ← (SYMBOLIC) QUAD

Affected by Flags: Principal Solution (−1)

QUAD

Remarks: QUAD calculates the second-degree Taylor series approximation of '*symb*$_1$' to convert it to quadratic form. The solution '*symb*$_2$' is exact if '*symb*$_1$' is second degree or less in *global*.

Since QUAD evaluates '*symb*$_1$', any variables in '*symb*$_1$' other than *global* should not exist in the current directory if they are to remain in the solution as formal variables.

QUAD generally does not work if *global* needs units to satisfy the equation.

Example: `'A*X^2+B*X+C=0' 'X' QUAD` returns
`'X=(-B+±1*√(B^2-4*(A*2/2)*C))/(2*(A*2/2))'`

which reduces to the familiar quadratic solution:

$$X = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

Related Commands: COLCT, EXPAN, ISOL, SHOW

QUOTE

Quote Argument Function: Returns its argument unevaluated.

{ }

Level 1	→	Level 1
'*symb*'	→	'*symb*'
obj	→	*obj*

Keyboard Access: ◀ SYMBOLIC NXT NXT QUOTE

Affected by Flags: None

Remarks: When an algebraic expression is evaluated, the arguments to a function in the expression are evaluated before the function. For example, when `'SIN(X)'` is evaluated, the name X is evaluated first, and the result is left on the stack as the argument for SIN.

This process creates a problem for functions that require symbolic arguments. For example, the function $\int$ requires as one of its arguments a name specifying the variable of integration. If evaluating an integral expression caused the name to be evaluated, the result of evaluation would be left on the stack for $\int$, rather than the name itself. To avoid this problem, the HP 48 automatically (and invisibly) quotes such arguments. When the quoted argument is evaluated, the unquoted argument is returned.

If a user-defined function takes symbolic arguments, those arguments must be quoted using QUOTE, as demonstrated in the following example.

Example: The following user-defined function *ArcLen* calculates the arc length of a function:

```
≪
 → start end expr var
 ≪
  start end
  expr var ∂ SQ 1 + √
  var ∫
 ≫
≫
```

ENTER ' ArcLen STO

To use this user-defined function in an algebraic expression, the symbolic arguments must be quoted:

```
'ArcLen(0,π,QUOTE(SIN(X)),QUOTE(X))'
```

Related Commands: APPLY, | (Where)

RAD

Radians Mode Command: Sets Radians angle mode.

Keyboard Access:

◀ RAD

◀ MODES ANGL RAD

Affected by Flags: None

Remarks: RAD sets flag −17 and clears flag −18, and displays the RAD annunciator.

In Radians angle mode, real-number arguments that represent angles are interpreted as radians, and real-number results that represent angles are expressed in radians.

Related Commands: DEG, GRAD

RAND

Random Number Command: Returns a pseudo-random number generated using a seed value, and updates the seed value.

Level 1	→	Level 1
	→	x_{random}

Keyboard Access: MTH NXT PROB RAND

Affected by Flags: None

Remarks: The HP 48 uses a linear congruential method and a seed value to generate a random number x_{random} in the range $0 \leq x < 1$. Each succeeding execution of RAND returns a value computed from a seed value based upon the previous RAND value. (Use RDZ to change the seed.)

Related Commands: COMB, PERM, RDZ, !

RANK

Matrix Rank Command: Returns the rank of a rectangular matrix.

{}

Level 1	→	Level 1
[[*matrix*]]	→	n_{rank}

Keyboard Access: (MTH) MATR MORE (NXT) RANK

Affected by Flags: Singular Value (−54)

Remarks: Rank is computed by calculating the singular values of the matrix and counting the number of nonnegligible values. If all computed singular values are zero, RANK returns zero. Otherwise RANK consults flag −54 as follows:

- If flag −54 is clear (the default), RANK counts all computed singular values that are less than or equal to 1.E−14 times the largest computed singular value.
- If flag −54 is set, RANK counts all nonzero computed singular values.

Related Commands: LQ, LSQ, QR

RANM

Random Matrix Command: Returns a matrix of specified dimensions that contains random integers in the range −9 through 9.

Level 1	→	Level 1
{ *m n* }	→	[[*random matrix*]]$_{m \times n}$
[[*matrix*]]$_{m \times n}$	→	[[*random matrix*]]$_{m \times n}$

Keyboard Access: (MTH) MATR MAKE RANM

RANM

Affected by Flags: None

Remarks: The probability of a particular nonzero digit occurring is 0.05; the probability of 0 occurring is 0.1.

Related Commands: RAND, RDZ

RATIO

Prefix Divide Function: Prefix form of / (divide) generated by the EquationWriter application.

{}

Level 2	Level 1	→	Level 1
z_1	z_2	→	$z_1 \ / \ z_2$
[*array*]	[[*matrix*]]	→	[[*array* × $matrix^{-1}$]]
[*array*]	*z*	→	[*array/z*]
z	'*symb*'	→	'*z/symb*'
'*symb*'	*z*	→	'*symb/z*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ / $symb_2$'
$\#n_1$	n_2	→	$\#n_3$
n_1	$\#n_2$	→	$\#n_3$
$\#n_1$	$\#n_2$	→	$\#n_3$
x_unit_1	y_unit_2	→	$(x/y)_unit_1 \ / unit_2$
x	*y_unit*	→	*(x/y)_1/unit*
x_unit	*y*	→	*(x/y)_unit*
'*symb*'	*x_unit*	→	'*symb/x_unit*'
x_unit	'*symb*'	→	'*x_unit/symb*'

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: RATIO is identical to / (divide), except that, in algebraic syntax, RATIO is a *prefix* function, while / is an *infix* function. For example, 'RATIO(A,2)' is equivalent to 'A/2'.

RATIO is generated internally by the EquationWriter application when ▲ is used to start a numerator. It provides no additional functionality to / and appears externally only in the string that the EquationWriter application leaves on the stack when → " " is pressed or when the calculator runs out of memory.

Related Commands: /

RCEQ

Recall from EQ Command: Returns the unevaluated contents of the reserved variable *EQ* from the current directory.

Level 1	→	Level 1
	→	obj_{EQ}

Keyboard Access:

← PLOT EQ

← PLOT NXT 3D EQ

← PLOT → EQ (program-entry mode)

← PLOT NXT 3D → EQ (program-entry mode)

Affected by Flags: None

Remarks: To recall the contents of *EQ* from a parent directory (when *EQ* doesn't exist in the current directory) evaluate the name *EQ*.

Related Commands: STEQ

RCI

Multiply Row by Constant Command: Multiplies row n of a matrix (or element n of a vector) by a constant x_{factor}, and returns the modified matrix.

{}

Level 3	Level 2	Level 1	→	Level 1
[[*matrix*]]$_1$	x_{factor}	$n_{\mathsf{rownumber}}$	→	[[*matrix*]]$_2$
[*vector*]$_1$	x_{factor}	$n_{\mathsf{elementnumber}}$	→	[*vector*]$_2$

Keyboard Access: (MTH) MATR ROW RCI

Affected by Flags: None

Remarks: RCI rounds the row number to the nearest integer, and treats vector arguments as column vectors.

Related Commands: RCIJ

RCIJ

Add Multiplied Row Command: Multiplies row i of a matrix by a constant x_{factor}, adds this product to row j of the matrix, and returns the modified matrix. Or, multiplies element i of a vector by a constant x_{factor}, adds this product to element j of the vector, and returns the modified vector.

{}

Level 4	Level 3	Level 2	Level 1	→	Level 1
[[*matrix*]]$_1$	x_{factor}	n_{rowi}	n_{rowj}	→	[[*matrix*]]$_2$
[*vector*]$_1$	x_{factor}	n_{elementi}	n_{elementj}	→	[*vector*]$_2$

Keyboard Access: (MTH) MATR ROW RCIJ

Affected by Flags: None

Remarks: RCIJ rounds the row numbers to the nearest integer, and treats vector arguments as column vectors.

Related Commands: RCI

RCL

Recall Command: Returns the unevaluated contents of a specified variable or plug-in object.

Level 1	→	Level 1
'*name*'	→	*obj*
PICT	→	*grob*
$:n_{port}$ $:n_{library}$	→	*obj*
$:n_{port}$ $:name_{backup}$	→	*obj*

Keyboard Access: → RCL

Affected by Flags: None

Remarks: RCL searches the entire current path, starting with the current directory. To search a different path, specify { *path name* }, where *path* is the new path to the variable *name*. The *path* subdirectory does not become the current subdirectory (unlike EVAL).

To recall a library or backup object, tag the library number or backup name with the appropriate port number (n_{port}), which must be an integer in the range 0 to 33. (A library can be recalled from RAM only.) Recalling a backup object brings a copy of its *contents* to the stack, not the entire backup object.

To search for a backup object, replace the port number with the wildcard character & , in which case the HP 48 will search (in order) ports 33 through 0, and the main memory for the named backup object.

Related Commands: STO

RCLALARM

Recall Alarm Command: Recalls a specified alarm.

{ }

Level 1	→	Level 1
n_{index}	→	{ *date time* obj_{action} x_{repeat} }

Keyboard Access: ◀ TIME ALRM RCLAL

Affected by Flags: None

Remarks: obj_{action} is the alarm execution action. If an execution action was not specified, obj_{action} defaults to an empty string.

x_{repeat} is the repeat interval in clock ticks, where 1 clock tick equals 1/8192 second. If a repeat interval was not specified, the default is 0.

Related Commands: DELALARM, FINDALARM, STOALARM

RCLF

Recall Flags Command: Returns a list containing two 64-bit binary integers representing the states of the 64 system and user flags, respectively.

Level 1	→	Level 1
	→	{ $\#n_{system}$ $\#n_{user}$ }

Keyboard Access: ◀ MODES FLAG NXT RCLF

Affected by Flags: Binary Integer Wordsize (−5 through −10)

The current wordsize must be 64 bits (the default wordsize) to recall the states of all 64 user flags and 64 system flags. If the current wordsize is 32, for example, RCLF returns two 32-bit binary integers.

Remarks: A bit with value 1 indicates that the corresponding flag is set; a bit with value 0 indicates that the corresponding flag is clear. The rightmost (least significant) bit of #n_{system} and #n_{user} indicate the states of system flag −1 and user flag +1, respectively.

Used with STOF, RCLF lets a program that alters the state of a flag or flags during program execution preserve the pre-program-execution flag status.

Related Commands: STOF

RCLKEYS

Recall Key Assignments Command: Returns the current user key assignments. This includes an S if the standard definitions are active (not suppressed) for those keys without user key assignments.

Level 1	→	Level 1
	→	{ obj_1 x_{key1} ⋯ obj_n x_{keyn} }
	→	{ S obj_1 x_{key1} ⋯ obj_n x_{keyn} }

Keyboard Access: ↩ MODES KEYS RCLK

Affected by Flags: User-Mode Lock (−61) and User Mode (−62) affect the status of the user keyboard.

Remarks: The argument x_{key} is a real number of the form $rc.p$ specifying the key by its row number r, its column number c, and its plane (shift) p. (For a definition of plane, see the entry for ASN.)

Related Commands: ASN, DELKEYS, STOKEYS

RCLMENU

Recall Menu Number Command: Returns the menu number of the currently displayed menu.

Level 1	→	Level 1
	→	x_{menu}

Keyboard Access: ◀ MODES MENU RCLM

Affected by Flags: None

Remarks: x_{menu} has the form *mm.pp*, where *mm* is the menu number and *pp* is the page of the menu. See the MENU entry for a list of the HP 48 built-in menus and the corresponding menu numbers.

Executing RCLMENU when the current menu is a user-defined menu (built by TMENU) returns `0.01` (in 2 Fix mode), indicating "Last menu".

Example: If the third page of the PRG DSPL menu is currently active, RCLMENU returns `13.03` (in 2 Fix mode).

Related Commands: MENU, TMENU

RCLΣ

Recall Sigma Command: Returns the current statistics matrix (the contents of reserved variable ΣDAT) from the current directory.

Level 1	→	Level 1
	→	*obj*

Keyboard Access:

◀ PLOT NXT ΣTHT ΣDAT

◀ STAT DATA ΣDAT

◀ PLOT NXT STAT ▶ ΣDAT (program-entry mode)

◀ STAT DATA ▶ ΣDAT (program-entry mode)

Affected by Flags: None

Remarks: To recall ΣDAT from a parent directory (when ΣDAT doesn't exist in the current directory), evaluate the name ΣDAT.

Related Commands: CLΣ, STOΣ, Σ+, Σ−

RCWS

Recall Wordsize Command: Returns the current wordsize in bits (1 through 64).

Level 1	→	Level 1
	→	*n*

Keyboard Access: MTH BASE NXT RCWS

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Related Commands: BIN, DEC, HEX, OCT, STWS

RDM

Redimension Array Command: Rearranges the elements of the argument according to specified dimensions.

Level 2	Level 1	→	Level 1
[*vector*]$_1$	{ n_{elements} }	→	[*vector*]$_2$
[*vector*]	{ n_{rows} m_{cols} }	→	[[*matrix*]]
[[*matrix*]]	{ n_{elements} }	→	[*vector*]
[[*matrix*]]$_1$	{ n_{rows} m_{cols} }	→	[[*matrix*]]$_2$
'*global*'	{ n_{elements} }	→	
'*global*'	{ n_{rows} m_{cols} }	→	

Keyboard Access: (MTH) MATR MAKE RDM

Affected by Flags: None

Remarks: If the list contains a single number n_{elements}, the result is an n-element vector. If the list contains two numbers n_{rows} and m_{cols}, the result is an $n \times m$ matrix.

Elements taken from the argument vector or matrix preserve the same row order in the resulting vector or matrix. If the result is dimensioned to contain fewer elements than the argument vector or matrix, excess elements from the argument vector or matrix at the end of the row order are discarded. If the result is dimensioned to contain more elements than the argument vector or matrix, the additional elements in the result at the end of the row order are filled with zeros (0,0 if the argument is complex).

If the argument vector or matrix is specified by *global*, the result replaces the argument as the contents of the variable.

Examples: [2 4 6 8] (2 2) RDM returns [[2 4][6 8]].

[[2 3 4][1 6 9]] 8 RDM returns [2 3 4 1 6 9 0 0].

Related Commands: TRN

RDZ

Randomize Command: Uses a real number $x_{\texttt{seed}}$ as a seed for the RAND command.

{ }

Level 1	→	Level 1
$x_{\texttt{seed}}$	→	

Keyboard Access: (MTH) (NXT) PROB RDZ

Affected by Flags: None

Remarks: If the argument is #0, a random value based on the system clock is used as the seed.

Related Commands: COMB, PERM, RAND, !

RE

Real Part Function: Returns the real part of the argument.

{ }

Level 1	→	Level 1
x	→	*x*
x_unit	→	*x*
(*x*, *y*)	→	*x*
[*R-array*]	→	[*R-array*]
[*C-array*]	→	[*R-array*]
'*symb*'	→	'RE(*symb*)'

Keyboard Access: (MTH) (NXT) CMPL RE

Affected by Flags: Numerical Results (−3)

RE

Remarks: If the argument is a vector or matrix, RE returns a real array, the elements of which are equal to the real parts of the corresponding elements of the argument array.

Related Commands: C→R, IM, R→C

RECN

Receive Renamed Object Command: Prepares the HP 48 to receive a file from another Kermit device, and to store the file in a specified variable.

{ }

Level 1	→	Level 1
'*name*'	→	
"*name*"	→	

Keyboard Access: ◀ I/O NXT RECN

Affected by Flags: I/O Device (−33), I/O Data Format (−35), RECV Overwrite (−36), I/O Messages (−39)

When an HP 48 sends an object, it automatically appends a header that tells a receiving HP 48 whether to use ASCII or binary mode. Flag −35 has an effect only if this header is not present.

Remarks: RECN is identical to RECV except that the name under which the received data is stored is specified in the stack.

Related Commands: BAUD, CKSM, CLOSEIO, FINISH, KERRM, KGET, PARITY, RECV, SEND, SERVER, TRANSIO

RECT

Rectangular Mode Command: Sets Rectangular coordinate mode.

Keyboard Access:

(MTH) VECTR (NXT) RECT

(MODES) ANGL RECT

Affected by Flags: None

Remarks: RECT clears flags −15 and −16, and clears the R∠Z and R∠∠ annuciators.

In Rectangular mode, vectors are displayed as rectangular components. Therefore, a 3D vector would appear as $[X\ Y\ Z]$.

Related Commands: CYLIN, SPHERE

RECV

Receive Object Command: Instructs the HP 48 to look for a named file from another Kermit device. The received file is stored in a variable named by the sender.

Keyboard Access: (I/O) RECV

Affected by Flags: I/O Device (−33), I/O Data Format (−35), RECV Overwrite (−36), I/O Messages (−39)

When an HP 48 sends an object, it automatically appends a header that tells a receiving HP 48 whether to use ASCII or binary mode. Flag −35 has an effect only if this header is not present.

Remarks: Since the HP 48does not normally look for incoming Kermit files, you must use RECV to tell it to do so.

Related Commands: BAUD, CKSM, FINISH, KGET, PARITY, RECN, SEND, SERVER, TRANSIO

REPEAT

REPEAT Command: Starts loop clause in WHILE ... REPEAT ... END indefinite loop structure.

See the WHILE entry for syntax information.

Keyboard Access: PRG BRCH WHILE REPEAT

Remarks: See the WHILE entry for more information.

Related Commands: END, WHILE

REPL

Replace Command: Replaces a portion of the level 3 target object with the level 1 object, beginning at a position specified in level 2.

Level 3	Level 2	Level 1	→	Level 1
[[*matrix*]]$_1$	$n_{position}$	[[*matrix*]]$_2$	→	[[*matrix*]]$_3$
[[*matrix*]]$_1$	{ n_{row} n_{column} }	[[*matrix*]]$_2$	→	[[*matrix*]]$_3$
[*vector*]$_1$	$n_{position}$	[*vector*]$_2$	→	[*vector*]$_3$
{ $list_{target}$ }	$n_{position}$	{ $list_1$ }	→	{ $list_{result}$ }
"$string_{target}$"	$n_{position}$	"$string_1$"	→	"$string_{result}$"
$grob_{target}$	{#*n* #*m*}	$grob_1$	→	$grob_{result}$
$grob_{target}$	(*x*,*y*)	$grob_1$	→	$grob_{result}$
PICT	{#*n* #*m*}	$grob_1$	→	
PICT	(*x*,*y*)	$grob_1$	→	

Keyboard Access:

◀ CHARS REPL

PRG LIST REPL

PRG GROB REPL

MTH MATR MAKE NXT REPL

Affected by Flags: None

Remarks: For arrays, $n_{position}$ counts in row order. For matrices, $n_{position}$ specifies the new location of the upper left-hand element of the replacement matrix.

For graphics objects, the upper left corner of $grob_1$ is positioned at the user-unit or pixel coordinates (x,y) or (#n #m). From there, it overwrites a rectangular portion of $grob_{target}$ or *PICT*. If $grob_1$ extends past $grob_{target}$ or *PICT* in either direction, it is truncated in that direction. If the specified coordinate is not on the target graphics object, the target graphics object does not change.

Examples: [[1 1 1 1] [1 1 1 1] [1 1 1 1]]
6 [[2 2] [2 2]] REPL returns
[[1 1 1 1] [1 2 2 1] [1 2 2 1]].

{ A B C D E } 2 { F G } REPL returns { A F G D E }.

"ABCDE" 5 "FG" REPL returns "ABCDFG".

ERASE PICT (0,0) # 5d # 5d BLANK NEG REPL replaces a portion of *PICT* with a 5 × 5 graphics object, each of whose pixels is on (dark), and whose upper left corner is positioned at (0,0) in *PICT*.

Related Commands: CHR, GOR, GXOR, NUM, POS, SIZE, SUB

RES

Resolution Command: Specifies the resolution of mathematical and statistical plots, where the resolution is the interval between values of the independent variable used to generate the plot.

{ }

Level 1	→	Level 1
$n_{interval}$	→	
$\#n_{interval}$	→	

Keyboard Access: ← PLOT FFAR RES

RES

Affected by Flags: None

Remarks: A real number n_{interval} specifies the interval in user units. A binary integer $\#n_{\mathsf{interval}}$ specifies the interval in pixels.

The resolution is stored as the fourth item in $PPAR$, with default value 0. The interpretation of the default value is summarized in the following table.

Plot Type	Default Interval
BAR	10 pixels (bar width = 10 pixel columns)
DIFFEQ	unlimited: step size is not constrained
FUNCTION	1 pixel (plots a point in every column of pixels)
CONIC	1 pixel (plots a point in every column of pixels)
TRUTH	1 pixel (plots a point in every column of pixels)
GRIDMAP	RES does not apply
HISTOGRAM	10 pixels (bin width = 10 pixel columns)
PARAMETRIC	[independent variable range in user units]/130
PARSURFACE	RES does not apply
PCONTOUR	RES does not apply
POLAR	2°, 2 grads, or $\pi/90$ radians
SCATTER	RES does not apply
SLOPEFIELD	RES does not apply
WIREFRAME	RES does not apply
YSLICE	1 pixel (plots a point in every column of pixels)

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

RESTORE

Restore HOME Command: Replaces the current *HOME* directory with the specified backup copy (:n_{port} :$name_{backup}$) previously created by ARCHIVE.

{ }

Level 1	→	Level 1
:n_{port} :$name_{backup}$	→	
backup	→	

Keyboard Access: ◀ MEMORY NXT RESTO

Affected by Flags: None

Remarks: The specified port number must be in the range 0 to 33. Ports 1 and 2 must be configured as independent RAM (see FREE).

To restore a *HOME* directory that was saved on a remote system using :IO:*name* ARCHIVE, put the backup object itself on the stack and execute RESTORE.

Example: To restore a *HOME* directory that was saved as the file *AUG1* on a remote system, execute `'AUG1' SEND` on the remote system, then execute the following on the HP 48:

```
RECV 'AUG1' RCL RESTORE
```

Related Commands: ARCHIVE

REVLIST

Reverse List Command: Reverses the order of the elements in a list.

Level 1	→	Level 1
{ obj_n ... obj_1 }	→	{ obj_1 ... obj_n }

Keyboard Access:

PRG LIST FROC REVLI

MTH LIST REVLI

Affected by Flags: None

Related Commands: ↑SORT

RKF

Solve for Initial Values (Runge-Kutta-Fehlberg) Command: Computes the solution to an initial value problem for a differential equation, using the Runge-Kutta-Fehlberg (4,5) method.

Level 3	Level 2	Level 1	→	Level 2	Level 1
{ *list* }	x_{tol}	x_{Tfinal}	→	{ *list* }	x_{tol}
{ *list* }	{ x_{tol} x_{hstep} }	x_{Tfinal}	→	{ *list* }	x_{tol}

Keyboard Access: ↶ SOLVE DIFFE RKF

Affected by Flags: None

Remarks: RKF solves y'(t)=f(t,y), where $y(t_0)=y_0$. The arguments and results are as follows:

- { *list* } contains three items in this order: the independent (t) and solution (y) variables, and the right-hand side of the differential equation (or a variable where the expression is stored).
- x_{tol} sets the absolute error tolerance. If a list is used, the first value is the absolute error tolerance and the second value is the initial candidate step size.
- x_{Tfinal} specifies the final value of the independent variable.

RKF repeatedly calls RKFSTEP as it steps from the initial value to x_{Tfinal}.

Example: Solve the following initial value problem for y(8), given that y(0) = 0:

$$y' = \frac{1}{(1+t^2)} - 2y^2 = f(t, y)$$

1. Store the independent variable's initial value, 0, in T.
2. Store the dependent variable's initial value, 0, in Y.
3. Store the expression, $\frac{1}{(1+t^2)} - 2y^2$, in F.
4. Enter a list containing these three items: { T Y F }.
5. Enter the tolerance. Use estimated decimal place accuracy as a guideline for choosing a tolerance: 0.00001.
6. Enter the final value for the independent variable: 8.

   The stack should look like this:

   ```
   { T Y F }
   .00001
   8
   ```

7. Press RKF. (The calculation takes a moment.) The variable T now contains 8, and Y now contains the value .123077277659.

The actual answer is .123076923077, so the calculated answer has an error of approximately .00000035, well within the specified tolerance.

Related Commands: RKFERR, RKFSTEP, RRK, RRKSTEP, RSBERR

RKFERR

Error Estimate for Runge-Kutta-Fehlberg Method Command:

Returns the absolute error estimate for a given step h when solving an initial value problem for a differential equation.

Level 2	Level 1	→	Level 4	Level 3	Level 2	Level 1
{ *list* }	*h*	→	{ *list* }	*h*	y_{delta}	*error*

Keyboard Access: ◀ SOLVE DIFFE RKFE

Affected by Flags: None

Remarks: The arguments and results are as follows:

- { *list* } contains three items in this order: the independent (t) and solution (y) variables, and the right-hand side of the differential equation (or a variable where the expression is stored).
- h is a real number that specifies the step.
- y_{delta} displays the change in solution for the specified step.
- *error* displays the absolute error for that step. A zero error indicates that the Runge-Kutta-Fehlberg method failed and that Euler's method was used instead.

The absolute error is the absolute value of the estimated error for a scalar problem, and the row (infinity) norm of the estimated error vector for a vector problem. (The latter is a bound on the maximum error of any component of the solution.)

Related Commands: RKF, RKFSTEP, RRK, RRKSTEP, RSBERR

RKFSTEP

Next Solution Step for RKF Command: Computes the next solution step (h_{next}) to an initial value problem for a differential equation.

Level 3	Level 2	Level 1	→	Level 3	Level 2	Level 1
{ *list* }	x_{tol}	h	→	{ *list* }	x_{tol}	h_{next}

Keyboard Access: ◀ SOLVE DIFFE RKFS

Affected by Flags: None

Remarks: The arguments and results are as follows:

- { *list* } contains three items in this order: the independent (t) and solution (y) variables, and the right-hand side of the differential equation (or a variable where the expression is stored).
- x_{tol} sets the tolerance value.
- h specifies the initial candidate step.
- h_{next} is the next candidate step.

The independent and solution variables must have values stored in them. RKFSTEP steps these variables to the next point upon completion.

Note that the actual step used by RKFSTEP will be less than the input value h if the global error tolerance is not satisfied by that value. If a stringent global error tolerance forces RKFSTEP to reduce its stepsize to the point that the Runge-Kutta-Fehlberg method fails, then RKFSTEP will use the Euler method to compute the next solution step and will consider the error tolerance satisfied. The Runge-Kutta-Fehlberg method will fail if the current independent variable is zero and the stepsize $\leq 1.3 \times 10^{-498}$ or if the variable is nonzero and the stepsize is 1.3×10^{-10} times its magnitude.

Related Commands: RKF, RKFERR, RRK, RRKSTEP, RSBERR

RL

Rotate Left Command: Rotates a binary integer one bit to the left.

{ }

Level 1	→	Level 1
#n_1	→	#n_2

Keyboard Access: (MTH) BASE (NXT) BIT RL

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The leftmost bit of #n_1 becomes the rightmost bit of #n_2.

Related Commands: RLB, RR, RRB

RLB

Rotate Left Byte Command: Rotates a binary integer one byte to the left.

{ }

Level 1	→	Level 1
#n_1	→	#n_2

Keyboard Access: (MTH) BASE (NXT) BYTE RLB

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The leftmost byte of #n_1 becomes the rightmost byte of #n_2. RLB is equivalent to executing RL eight times.

Related Commands: RL, RR, RRB

RND

Round Function: Rounds an object to a specified number of decimal places or significant digits, or to fit the current display format.

{ }

Level 2	Level 1	→	Level 1
z_1	n_{round}	→	z_2
z	'$symb_{round}$'	→	'RND(z,$symb_{round}$)'
'$symb$'	n_{round}	→	'RND($symb$,n_{round})'
'$symb_1$'	'$symb_{round}$'	→	'RND($symb_1$,$symb_{round}$)'
[$array_1$]	n_{round}	→	[$array_2$]
x_unit	n_{round}	→	y_unit
x_unit	'$symb_{round}$'	→	'RND(x_unit,$symb_{round}$)'

Keyboard Access: (MTH) REAL (NXT) (NXT) RND

Affected by Flags: Numerical Results (−3)

Remarks: n_{round} (or $symb_{round}$ if flag −3 is set) controls how the level 2 argument is rounded, as follows:

n_{round} or $symb_{round}$	Effect on Level 2 Argument
0 through 11	Rounded to n decimal places.
−1 through −11	Rounded to n significant digits.
12	Rounded to the current display format.

For complex numbers and arrays, each real number element is rounded. For unit objects, the numerical part of the object is rounded.

Examples: (4.5792,8.1275) 2 RND returns (4.58,8.13).

[2.34907 3.96351 2.73453] -2 RND returns [2.3 4 2.7].

Related Commands: TRNC

RNRM

Row Norm Command: Returns the row norm (infinity norm) of its argument array.

Level 1	→	Level 1
[*array*]	→	$x_{row\,norm}$

Keyboard Access: (MTH) MATR NORM RNRM

Affected by Flags: None

Remarks: The row norm is the maximum (over all rows) of the sums of the absolute values of all elements in each row. For a vector, the row norm is the largest absolute value of any of its elements.

Related Commands: CNRM, CROSS, DET, DOT

ROLL

Roll Objects Command: Moves the contents of a specified level to level 1, and rolls upwards the portion of the stack beneath the specified level.

Level n+1 ... Level 2	Level 1	→	Level n ... Level 2	Level 1
obj_n ... obj_1	*n*	→	obj_{n-1} ... obj_1	obj_n

Keyboard Access: ↰ (STACK) ROLL

Affected by Flags: None

Remarks: 3 ROLL is equivalent to ROT.

Related Commands: OVER, PICK, ROLLD, ROT, SWAP

ROLLD

Roll Down Command: Moves the contents of level 1 to a specified level, and rolls downwards the portion of the stack beneath the specified level.

Lvl n+1 ... Lvl 2	Lvl 1	→	Lvl n	Lvl n-1 ... Lvl 1
obj_n ... obj_1	n	→	obj_1	obj_n ... obj_2

Keyboard Access: ↰ STACK ROLLD

Affected by Flags: None

Related Commands: OVER, PICK, ROLL, ROT, SWAP

ROOT

Root-Finder Command: Returns a real number x_{root} that is a value of the specified variable *global* for which the specified program or algebraic object most nearly evaluates to zero or a local extremum.

Level 3	Level 2	Level 1	→	Level 1
≪ *program* ≫	'*global*'	*guess*	→	x_{root}
≪ *program* ≫	'*global*'	{ *guesses* }	→	x_{root}
'*symb*'	'*global*'	*guess*	→	x_{root}
'*symb*'	'*global*'	{ *guesses* }	→	x_{root}

Keyboard Access: ↰ SOLVE ROOT ROOT

Affected by Flags: None

Remarks: ROOT is the programmable form of the HP Solve application.

ROOT

guess is an initial estimate of the solution. ROOT produces an error if it cannot find a solution, returning the message `Bad Guess(es)` if one or more of the guesses lie outside the domain of the equation, or returns the message `Constant?` if the equation returns the same value at every sample point. ROOT does *not* return interpretive messages when a root is found.

ROT

Rotate Objects Command: Rotates the first three objects on the stack, moving the object in level 3 to level 1.

Level 3	Level 2	Level 1	→	Level 3	Level 2	Level 1
obj_3	obj_2	obj_1	→	obj_2	obj_1	obj_3

Keyboard Access: ◀ STACK ROT

Affected by Flags: None

Remarks: ROT is equivalent to `3 ROLL`.

Related Commands: OVER, PICK, ROLL, ROLLD, SWAP

→ROW

Matrix to Rows Command: Transforms a matrix into a series of row vectors and returns the vectors and a row count, or transforms a vector into its elements and returns the elements and an element count.

{ }

Level 1	→	Level n+1 ...	Level 2	Level 1
[[*matrix*]]	→	[*vector*]$_{row\ 1}$	[*vector*]$_{row\ n}$	$n_{rowcount}$
[*vector*]	→	$element_1$	$element_n$	$n_{elementcount}$

Keyboard Access: (MTH) MATR ROW +ROW

Affected by Flags: None

Related Commands: →COL, COL→, ROW→

ROW+

Insert Row Command: Inserts an array into a matrix (or one or more numbers into a vector) at the position indicated by n_{index}, and returns the modified matrix (or vector).

{ }

Level 3	Level 2	Level 1	→	Level 1
[[*matrix*]]$_1$	[*matrix*]$_2$	n_{index}	→	[[*matrix*]]$_3$
[[*matrix*]]$_1$	[*vector*]$_{row}$	n_{index}	→	[[*matrix*]]$_2$
[*vector*]$_1$	$n_{element}$	n_{index}	→	[*vector*]$_2$

Keyboard Access: (MTH) MATR ROW ROW+

Affected by Flags: None

Remarks: The inserted array must have the same number of columns as the target array.

n_{index} is rounded to the nearest integer. The original array is redimensioned to include the new columns or elements, and the elements at and below the insertion point are shifted down.

Related Commands: COL−, COL+, ROW−, RSWP

ROW−

Delete Row Command: Deletes row n of a matrix (or element n of a vector), and returns the modified matrix (or vector) and the deleted row (or element).

{ }

Level 2	Level 1	→	Level 2	Level 1
[[*matrix*]]$_1$	n_{row}	→	[[*matrix*]]$_2$	[*vector*]$_{\text{row}}$
[*vector*]$_1$	n_{element}	→	[*vector*]$_2$	*element*$_n$

Keyboard Access: (MTH) MATR ROW ROW−

Affected by Flags: None

Remarks: n_{row} or n_{element} is rounded to the nearest integer.

Related Commands: COL−, COL+, ROW+, RSWP

ROW→

Rows to Matrix Command: Transforms a series of row vectors and a row count into a matrix containing those rows, or transforms a sequence of numbers and an element count into a vector with those numbers as elements.

Level$_{n+1}$ ...	Level 2	Level 1	→	Level 1
[*vector*]$_{\text{row 1}}$	[*vector*]$_{\text{row n}}$	n_{rowcount}	→	[[*matrix*]]
element$_1$	*element*$_n$	$n_{\text{elementcount}}$	→	[*vector*]$_{\text{column}}$

Keyboard Access: (MTH) MATR ROW ROW+

Affected by Flags: None

Related Commands: →COL, COL→, →ROW

RR

Rotate Right Command: Rotates a binary integer one bit to the right.

{ }

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Keyboard Access: (MTH) BASE (NXT) BIT RR

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The rightmost bit of $\#n_1$ becomes the leftmost bit of $\#n_2$.

Related Commands: RL, RLB, RRB

RRB

Rotate Right Byte Command: Rotates a binary integer one byte to the right.

{ }

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Keyboard Access: (MTH) BASE (NXT) BYTE RRB

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12).

Remarks: The rightmost byte of $\#n_1$ becomes the leftmost byte of $\#n_2$. RRB is equivalent to doing RR eight times.

Related Commands: RL, RLB, RR

RREF

Reduced Row Echelon Form Command: Converts a rectangular matrix to reduced row echelon form.

{}

Level 1	→	Level 1
[[*matrix*]]$_A$	→	[[*matrix*]]$_R$

Keyboard Access: (MTH) MATR FACTR RREF

Affected by Flags: Singular Values (−54)

Remarks: Converts a given matrix into reduced row echelon form. Since row echelon form is primarly used for studying systems of linear equations, RREF ignores very small pivots if system flag −54 is clear.

Related Commands: LU

RRK

Solve for Initial Values (Rosenbrock, Runge-Kutta) Command: Computes the solution to an intial value problem for a differential equation with known partial derivatives.

Level 3	Level 2	Level 1	→	Level 2	Level 1
{ *list* }	x_{tol}	x_{Tfinal}	→	{ *list* }	x_{tol}
{ *list* }	{ x_{tol} x_{hstep} }	x_{Tfinal}	→	{ *list* }	x_{tol}

Keyboard Access: (↩) (SOLVE) DIFFE RRK

Affected by Flags: None

Remarks: RRK solves y'(t)=f(t,y), where $y(t_0)=y_0$. The arguments and results are as follows:

- { *list* } contains five items in this order:
  - The independent variable (t).
  - The solution variable (y).
  - The right-hand side of the differential equation (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the solution variable) (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the independent variable (or a variable where the expression is stored).
- x_{tol} sets the tolerance value. If a list is used, the first value is the tolerance and the second value is the initial candidate step size.
- x_{Tfinal} specifies the final value of the independent variable.

RRK repeatedly calls RKFSTEP as it steps from from the initial value to x_{Tfinal}.

Example: Solve the following initial value problem for y(8), given that y(0) = 0:

$$y' = \frac{1}{(1+t^2)} - 2y^2 = f(t,y)$$

The derivative of the function with respect to y ($\partial f/\partial y$) is -4y, and the derivative of the function with respect to t ($\partial f/\partial t$) is $\frac{-2t}{(1+t^2)^2}$.

1. Store the independent variable's initial value, 0, in T.
2. Store the dependent variable's initial value, 0, in Y.
3. Store the expression, $\frac{1}{(1+t^2)} - 2y^2$, in F.
4. Store $\partial f/\partial y$, -4y, in FY.
5. Store $\partial f/\partial t$, $\frac{-2t}{(1+t^2)^2}$, in FT.
6. Enter these five items in a list: { T Y F FY FT }.
7. Enter the tolerance. Use estimated decimal place accuracy as a guideline for choosing a tolerance: 0.00001.
8. Enter the final value for the independent variable: 8.

The stack should look like this:

```
{ T Y F FY FT}
.00001
8
```

9. Press FFK. (The calculation takes a moment.) The variable T now contains 8, and Y now contains the value .123077277659.

The actual answer is .123076923077, so the calculated answer has an error of approximately .00000035, well within the specified tolerance.

Related Commands: RKF, RKFERR, RKFSTEP, RRKSTEP, RSBERR

RRKSTEP

Next Solution Step and Method (RKF or RRK) Command:
Computes the next solution step (h_{next}) to an intial value problem for a differential equation, and displays the method used to arrive at that result.

Lvl 4	Lvl 3	Lvl 2	Lvl 1	→	Lvl 4	Lvl 3	Lvl 2	Lvl 1
{ *list* }	x_{tol}	*h*	*last*	→	{ *list* }	x_{tol}	h_{next}	*current*

Keyboard Access: ↩ SOLVE DIFFE RRKS

Affected by Flags: None

Remarks: The arguments and results are as follows:

- { *list* } contains five items in this order:
  - The independent variable (t).
  - The solution variable (y).
  - The right-hand side of the differential equation (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the solution variable) (or a variable where the expression is stored).

- □ The partial derivative of y'(t) with respect to the independent variable (or a variable where the expression is stored).

- x_{tol} is the tolerance value.
- h specifies the initial candidate step.
- *last* specifies the last method used (RKF = 1, RRK = 2). If this is the first time you are using RRKSTEP, enter 0.
- *current* displays the current method used to arrive at the next step.
- h_{next} is the next candidate step.

The independent and solution variables must have values stored in them. RRKSTEP steps these variables to the next point upon completion.

Note that the actual step used by RRKSTEP will be less than the input value h if the global error tolerance is not satisfied by that value. If a stringent global error tolerance forces RRKSTEP to reduce its stepsize to the point that the Runge-Kutta-Fehlberg or Rosenbrock methods fails, then RRKSTEP will use the Euler method to compute the next solution step and will consider the error tolerance satisfied. The Rosenbrock method will fail if the current independent variable is zero and the stepsize $\leq 2.5 \times 10^{-499}$ or if the variable is nonzero and the stepsize is 2.5×10^{-11} times its magnitude. The Runge-Kutta-Fehlberg method will fail if the current independent variable is zero and the stepsize $\leq 1.3 \times 10^{-498}$ or if the variable is nonzero and the stepsize is 1.3×10^{-10} times its magnitude.

Related Commands: RKF, RKFERR, RKFSTEP, RRK, RSBERR

RSBERR

Error Estimate for Rosenbrock Method Command: Returns an error estimate for a given step h when solving an initial values problem for a differential equation.

Level 2	Level 1	→	Level 4	Level 3	Level 2	Level 1
{ *list* }	*h*	→	{ *list* }	*h*	y_{delta}	*error*

Keyboard Access: ◀ SOLVE DIFFE RSBER

Affected by Flags: None

Remarks: The arguments and results are as follows:

- { *list* } contains five items in this order:
  - The independent variable (t).
  - The solution variable (y).
  - The right-hand side of the differential equation (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the solution variable) (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the independent variable (or a variable where the expression is stored).
- h is a real number that specifies the initial step.
- y_{delta} displays the change in solution.
- *error* displays the absolute error for that step. The *absolute* error is the absolute value of the estimated error for a scalar problem, and the row (infinity) norm of the estimated error vector for a vector problem. (The latter is a bound on the maximum error of any component of the solution.) A zero error indicates that the Rosenbrock method failed and Euler's method was used instead.

Related Commands: RKF, RKFERR, RKFSTEP, RRK, RRKSTEP

RSD

Residual Command: Computes the residual **B** − **AZ** of the arrays **B**, **A**, and **Z**.

{}

Level 3	Level 2	Level 1	→	Level 1
[*vector*]$_B$	[[*matrix*]]$_A$	[*vector*]$_Z$	→	[*vector*]$_{B\ -A\ Z}$
[[*matrix*]]$_B$	[[*matrix*]]$_A$	[[*matrix*]]$_Z$	→	[[*matrix*]]$_{B\ -A\ Z}$

Keyboard Access: (MTH) MATR (NXT) RSD

Affected by Flags: None

Remarks: **A**, **B**, and **Z** are restricted as follows:

- **A** must be a matrix.
- The number of columns of **A** must equal the number of elements of **Z** if **Z** is a vector, or the number of rows of **Z** if **Z** is a matrix.
- The number of rows of **A** must equal the number of elements of **B** if **B** is a vector, or the number of rows of **B** if **B** is a matrix.
- **B** and **Z** must both be vectors or both be matrices.
- **B** and **Z** must have the same number of columns if they are matrices.

RSD is typically used for computing a correction to **Z**, where **Z** has been obtained as an approximation to the solution **X** to the system of equations $\mathbf{AX} = \mathbf{B}$.

RSWP

Row Swap Command: Swaps rows i and j of a matrix and returns the modified matrix, or swaps elements i and j of a vector and returns the modified vector.

{ }

Level 3	Level 2	Level 1	→	Level 1
[[*matrix*]]$_1$	$n_{\text{row i}}$	$n_{\text{row j}}$	→	[[*matrix*]]$_2$
[*vector*]$_1$	$n_{\text{element i}}$	$n_{\text{element j}}$	→	[*vector*]$_2$

Keyboard Access: (MTH) MATR ROW (NXT) RSWP

Affected by Flags: None

Remarks: Row numbers are rounded to the nearest integer. Vector arguments are treated as column vectors.

Related Commands: CSWP, ROW+, ROW−

R→B

Real to Binary Command: Converts a positive real integer to its binary integer equivalent.

{ }

Level 1	→	Level 1
n	→	#*n*

Keyboard Access: (MTH) BASE R→B

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: For any value of $n \leq 0$, the result is # 0. For any value of $n \geq 1.8446744073709E19$ (base 10), the result is # FFFFFFFFFFFFFFFF (base 16).

Related Commands: B→R

R→C

Real to Complex Command: Combines two real numbers or real arrays into a single complex number or complex array.

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	(*x*,*y*)
[*R-array*$_1$]	[*R-array*$_2$]	→	[*C-array*]

Keyboard Access:

PRG TYPE NXT R→C

MTH NXT CMPL R→C

Affected by Flags: None

Remarks: The level 2 argument represents the real element(s) of the complex result. The level 1 argument represents the imaginary element(s) of the complex result.

Array arguments must have the same dimensions.

Related Commands: C→R, IM, RE

R→D

Radians to Degrees Function: Converts a real number expressed in radians to its equivalent in degrees.

Level 1	→	Level 1
x	→	(180/π) *x*
'*symb*'	→	'R→D(*symb*)'

Keyboard Access: MTH REAL NXT NXT R→D

Affected by Flags: Numerical Results (−3)

Remarks: This function operates independently of the angle mode.

Related Commands: D→R

SAME

Same Object Command: Compares two objects, and returns a true result (1) if they are identical, and a false result (0) if they are not.

Level 2	Level 1	→	Level 1
obj_1	obj_2	→	0/1

Keyboard Access: PRG TEST NXT SAME

Affected by Flags: None

Remarks: SAME is identical in effect to == for all object types except algebraics, names, and some units. (For algebraics and names, == returns an expression that can be evaluated to produce a test result based on numerical values.)

Examples: « A B » (4,5) SAME returns 0.

{ A B } { B A } SAME returns 0.

"CATS" "CATS" SAME returns 1.

Related Commands: TYPE, ==

SBRK

Serial Break Command: Interrupts serial transmission or reception.

Keyboard Access: ◀ I/O NXT SERIA SBRK

Affected by Flags: I/O Device (−33)

Remarks: SBRK is typically used when a problem occurs in a serial data transmission.

Related Commands: BUFLEN, SRECV, STIME, XMIT

SCALE

Scale Plot Command: Adjusts the first two parameters in *PPAR*, $\{ x_{min}, y_{min} \}$ and $\{ x_{max}, y_{max} \}$, so that x_{scale} and y_{scale} are the new plot horizontal and vertical scales, and the center point doesn't change.

{ }

Level 2	Level 1	→	Level 1
x_{scale}	y_{scale}	→	

Keyboard Access: ◀ PLOT PPAR NXT SCALE

Affected by Flags: None

Remarks: The scale in either direction is the number of user units per tick mark. The default scale in both directions is 1 user unit per tick mark.

SCALE

Related Commands: AUTO, CENTR, *H, *W

SCATRPLOT

Draw Scatter Plot Command: Draws a scatterplot of (x, y) data points from the specified columns of the current statistics matrix (reserved variable ΣDAT).

Keyboard Access: ↰ STAT PLOT SCATR

Affected by Flags: None

Remarks: The data columns plotted are specified by XCOL and YCOL, and are stored as the first two parameters in the reserved variable ΣPAR. If no data columns are specified, columns 1 (independent) and 2 (dependent) are selected by default. The y-axis is autoscaled and the plot type is set to SCATTER.

When SCATRPLOT is executed from a program, the resulting display does not persist unless PICTURE or PVIEW is subsequently executed.

If PICTURE *is* subsequently executed, pressing STATL in the Picture environment draws a line to fit the data using the currently specified statistical model.

Example: The following program plots a scatter plot of the data in columns 3 and 4 of ΣDAT, draws a best fit line, and displays the plot:

```
« 3 XCOL 4 YCOL SCATRPLOT BESTFIT ΣLINE STEQ
  FUNCTION DRAW { # 0d # 0d } PVIEW 7 FREEZE »
```

Related Commands: BARPLOT, PICTURE, HISTPLOT, PVIEW, SCLΣ, XCOL, YCOL

SCATTER

Scatter Plot Type Command: Sets the plot type to SCATTER.

Keyboard Access: ◀ PLOT NXT SCATT PTYPE SCATT

Affected by Flags: None

Remarks: When the plot type is SCATTER, the DRAW command plots points by obtaining x and y coordinates from two columns of the current statistics matrix (reserved variable ΣDAT). The columns are specified by the first and second parameters in the reserved variable ΣPAR (using the XCOL and YCOL commands). The plotting parameters are specified in the reserved variable $PPAR$, which has this form:

{ ($x_{\mathbf{min}}$, $y_{\mathbf{min}}$) ($x_{\mathbf{max}}$, $y_{\mathbf{max}}$) *indep res axes ptype depend* }

For plot type SCATTER, the elements of $PPAR$ are used as follows:

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of $PICT$ (the lower left corner of the display range). The default value is (-6.5,-3.1).
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of $PICT$ (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a name specifying the independent variable. The default value of *indep* is X.
- *res* is not used.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command SCATTER places the name SCATTER in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is Y.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE,

SCATTER

PCONTOUR, POLAR, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

SCHUR

Schur Decomposition of a Square Matrix Command: Returns the Schur decomposition of a square matrix.

{ }

Level 1	→	Level 2	Level 1
[[*matrix*]]$_A$	→	[[*matrix*]]$_Q$	[[*matrix*]]$_T$

Keyboard Access: (MTH) MATR FACTR SCHUR

Affected by Flags: None

Remarks: SCHUR decomposes A into two matrices Q and T:

- If A is a complex matrix, Q is a unitary matrix, and T is an upper-triangular matrix.
- If A is a real matrix, Q is an orthogonal matrix, and T is an upper quasi-triangular matrix (T is upper block triangular with 1×1 or 2×2 diagonal blocks where the 2×2 blocks have complex conjugate eigenvalues).

In either case, $A \cong Q\times T\times\mathrm{TRN}(Q)$.

Related Commands: LQ, LU, QR, SVD, SVL, TRN

SCI

Scientific Mode Command: Sets the number display format to Scientific mode, which displays one digit to the left of the fraction mark and n significant digits to the right.

{ }

Level 1	→	Level 1
n	→	

Keyboard Access: ◀ MODES FMT SCI

Affected by Flags: None

Remarks: Scientific mode is equivalent to scientific notation using $n + 1$ significant digits, where $0 \leq n \leq 11$. (Values for n outside this range are rounded to the nearest integer.) In Scientific mode, numbers are displayed and printed like this:

(sign) mantissa E *(sign) exponent*

where the mantissa has the form $n.(n \ldots)$ and has zero to 11 decimal places, and the exponent has one to three digits.

Example: The number 103.6 in Scientific mode to four decimal places appears as `1.0360E2`.

Related Commands: ENG, FIX, STD

SCLΣ

Scale Sigma Command: Adjusts (x_{min}, y_{min}) and (x_{max}, y_{max}) in *PPAR* so that a subsequent scatter plot exactly fills *PICT*.

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: When the plot type is SCATTER, the command AUTO incorporates the functions of SCLΣ. In addition, the command

SCLΣ

SCATRPLOT automatically executes AUTO to achieve the same result. SCLΣ is included in the HP 48 for compatibility with the HP 28.

Related Commands: AUTO, SCATRPLOT

SCONJ

Store Conjugate Command: Conjugates the contents of a named object.

{ }

Level 1	→	Level 1
'*name*'	→	

Keyboard Access: ◀ MEMORY ARITH NXT SCONJ

Affected by Flags: None

Remarks: The named object must be a number, an array, or an algebraic object. For information on conjugation, see CONJ.

Related Commands: CONJ, SINV, SNEG

SDEV

Standard Deviation Command: Calculates the sample standard deviation of each of the m columns of coordinate values in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{sdev}
	→	$[\ x_{sdev1}\ x_{sdev2}\ \cdots\ x_{sdevm}\ ]$

Keyboard Access: ◀ STAT 1WAR SDEV

Affected by Flags: None

Remarks: SDEV returns a vector of m real numbers, or a single real number if $m = 1$. The standard deviations (the square root of the variances) are computed using this formula:

$$\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

where x_i is the ith coordinate value in a column, $\overline{x}$ is the mean of the data in this column, and n is the number of data points.

Related Commands: MAXΣ, MEAN, MINΣ, PSDEV, PVAR, TOT, VAR

SEND

Send Object Command: Sends a copy of the named objects to a Kermit device.

Level 1	→	Level 1
'name'	→	
{ $name_1$ ... $name_n$ }	→	
{{ $name_{old}$ $name_{new}$ } *name* ... }	→	

Keyboard Access: ◀ I/O SEND

Affected by Flags: I/O Device (−33), I/O Data Format (−35), I/O Messages (−39)

If flag −35 is clear (ASCII transfer), the translation setting also has an effect.

Remarks: Data is always sent from a local Kermit, but can be sent either to another local Kermit (which must execute RECV or RECN) or to a server Kermit.

SEND

To rename an object when sending it, include the old and new names in an embedded list.

Examples: Executing `{{ AAA BBB }} SEND` sends the variable named *AAA* but changes its name to *BBB*.

Executing `{{ AAA BBB } CCC } SEND` sends *AAA* as *BBB* and sends *CCC* under its own name. (If the new name is not legal on the HP 48, just enter it as a string.)

Related Commands: BAUD, CLOSEIO, CKSM, FINISH, KERRM, KGET, PARITY, RECN, RECV, SERVER, TRANSIO

SEQ

Sequential Calculation Command: Returns a list of results generated by repeatedly executing obj_{exec} using *index* over the range x_{start} to x_{end}, in increments of x_{incr}.

{ }

Level 5	Level 4	Level 3	Level 2	Level 1	→	Level 1
obj_{exec}	*index*	x_{start}	x_{end}	x_{incr}	→	{ *list* }

Keyboard Access: (PRG) LIST PROC (NXT) SEQ

Affected by Flags: None

Remarks: obj_{exec} is nominally a program or algebraic object that is a function of *index*, but can actually be any object. *index* must be a global or local name. The remaining objects can be anything that will evaluate to real numbers.

The action of SEQ for arbitrary inputs can be predicted exactly from this equivalent program:

$$x_{start}\ x_{end}\ \text{FOR}\ index\ obj_{exec}\ \text{EVAL}\ x_{incr}\ \text{STEP}\ n \rightarrow \text{LIST}$$

where n is the number of new objects left on the stack by the FOR ... STEP loop. Notice that *index* becomes a local variable regardless of its original type.

Example: 'n^2' 'n' 1 4 1 returns { 1 4 9 16 }.

Related Commands: DOSUBS, STREAM

SERVER

Server Mode Command: Selects Kermit Server mode.

Keyboard Access:



Affected by Flags: I/O Device (−33), I/O Data Format (−35), RECV Overwrite (−36), I/O Messages (−39)

Remarks: A Kermit server (a Kermit device in Server mode) passively processes requests sent to it by the local Kermit. The server receives data in response to SEND, transmits data in response to KGET, terminates Server mode in response to FINISH or LOGOUT, and transmits a directory listing in response to a generic directory request.

Server mode supports Kermit Host Command packets. This allows you, for instance, to use a PC to type into the HP 48's command line. (This is especially convenient while testing programs.) Do this as follows:

1. Set up the HP 48 for data transfer to a computer, as described in "Transferring Data Between the HP 48 and a Computer" in chapter 27 of the *HP 48 User's Guide*.
2. Execute SERVER to set the HP 48 to Server mode.
3. On the PC, type REMOTE HOST followed by up to 89 characters to be entered into the HP 48 command line.
4. Press Return to transmit and execute the commands. The HP 48 executes the transmitted commands, then sends back to the PC's display the resulting contents of the stack as the HP 48 would normally display them.

SERVER

If you use a PC to write programs for the HP 48, you should include the *%%HP...;* header in the program. See the discussion of ASCII mode in chapter 27 of the *HP 48 User's Guide*.

Related Commands: BAUD, CKSM, FINISH, KERRM, KGET, PARITY, PKT, RECN, RECV, SEND, TRANSIO

SF

Set Flag Command: Sets a specified user or system flag.

{ }

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	

Keyboard Access:

PRG TEST NXT NXT SF

◀ MODES FLAG SF

Affected by Flags: None

Remarks: User flags are numbered 1 through 64. System flags are numbered −1 through −64. See appendix C, "System Flags," for a listing of HP 48 system flags and their flag numbers.

Related Commands: CF, FC?, FC?C, FS?, FS?C

SHOW

Show Variable Command: Returns '$symb_2$', which is equivalent to '$symb_1$' except that all implicit references to a variable *name* are made explicit.

Level 2	Level 1	→	Level 1
'$symb_1$'	'*name*'	→	'$symb_2$'
'$symb_1$'	{ $name_1$ $name_2$ ... }	→	'$symb_2$'

Keyboard Access: ◀ SYMBOLIC SHOW

Affected by Flags: Numerical Results (−3)

Remarks: If the level 1 argument is a list, SHOW evaluates all global variables in '$symb_1$' *not* contained in the list.

Example: If 7 is stored in C and 5 is stored in D, then

```
'X-Y+2*C+3*D' { X Y } SHOW
```

returns 'X-Y+14+15'.

Related Commands: COLCT, EXPAN, ISOL, QUAD

SIDENS

Silicon Intrinsic Density Command: Calculates the intrinsic density of silicon as a function of temperature, x_T.

{ }

Level 1	→	Level 1
x_T	→	$x_{density}$
x_unit	→	*x_1/cm*^{3}
'*symb*'	→	'SIDENS(*symb*)'

SIDENS

Keyboard Access: ◀ EQ LIB UTILS SIDEN

Affected by Flags: Numerical Results (−3)

Remarks: If x_T is a unit object, it must reduce to a pure temperature, and the density is returned as a unit object with units of $1/cm^3$.

If x_T is a real number, its units are assumed to be K, and the density is returned as a real number with implied units of $1/cm^3$.

x_T must be between 0 and 1685 K.

SIGN

Sign Function: Returns the sign of a real number argument, the sign of the numerical part of a unit object argument, or the unit vector in the direction of a complex number argument.

{ }

Level 1	→	Level 1
z_1	→	z_2
x_unit	→	x_{sign}
'*symb*'	→	'SIGN(*symb*)'

Keyboard Access:

MTH REAL NXT SIGN (returns the sign of a number)

MTH NXT CMPL NXT SIGN (returns the unit vector of a complex number)

Affected by Flags: Numerical Results (−3)

Remarks: For real number and unit object arguments, the sign is defined as +1 for positive arguments, −1 for negative arguments, and 0 for argument 0.

For a complex argument:

$$\text{SIGN}(x+iy) = \frac{x}{\sqrt{x^2+y^2}} + \frac{iy}{\sqrt{x^2+y^2}}$$

Examples: 32_ft SIGN returns 1.

(1,1) SIGN returns (.707106781187,.707106781187).

Related Commands: ABS, MANT, XPON

SIMU

Simultaneous Plotting Command: Enables and disables simultaneous plotting.

Keyboard Access: ◀ PLOT NXT FLAG SIMU

Affected by Flags: Simultaneous Plotting (−28)

Remarks: SIMU changes to SIMU■ when flag −28 is enabled (and simultaneous plotting is enabled).

If the calculator is in program entry mode, pressing the menu key echoes AXES, CNCT, and SIMU flag numbers to the command line. Pressing ◀ or ▶ first echoes the flag numbers and SF or CF to the command line.

Related Commands: AXES, CF, SF

SIN

Sine Analytic Function: Returns the sine of the argument.

{ }

Level 1	→	Level 1
z	→	sin *z*
'*symb*'	→	'SIN(*symb*)'
$x_unit_{angular}$	→	sin ($x_unit_{angular}$)

SIN

Keyboard Access: (SIN)

Affected by Flags: Numerical Results (−3), Angle Mode (−17, −18)

Remarks: For real arguments, the current angle mode determines the number's units, unless angular units are specified.

For complex arguments, $\sin(x + iy) = \sin x \cosh y + i \cos x \sinh y$.

If the argument for SIN is a unit object, then the specified angular unit overrides the angle mode to determine the result. Integration and differentiation, on the other hand, always observe the angle mode. Therefore, to correctly integrate or differentiate expressions containing SIN with a unit object, the angle mode must be set to Radians (since this is a "neutral" mode).

Related Commands: ASIN, COS, TAN

SINH

Hyperbolic Sine Analytic Function: Returns the hyperbolic sine of the argument.

{ }

Level 1	→	Level 1
z	→	sinh *z*
'*symb*'	→	'SINH(*symb*)'

Keyboard Access: (MTH) HYP SINH

Affected by Flags: Numerical Results (−3)

Remarks: For complex arguments,
$\sinh(x + iy) = \sinh x \cos y + i \cosh x \sin y$.

Related Commands: ASINH, COSH, TANH

SINV

Store Inverse Command: Replaces the contents of the named variable with its inverse.

{ }

Level 1	→	Level 1
'*name*'	→	

Keyboard Access: ◀ MEMORY HRITH NXT SINV

Affected by Flags: None

Remarks: The named object must be a number, a matrix, an algebraic object, or a unit object. For information on reciprocals, see INV.

Related Commands: INV, SCONJ, SNEG

SIZE

Size Command: Returns the number of characters in a string, the number of elements in a list, the dimensions of an array, the number of objects in a unit object or algebraic object, or the dimensions of a graphics object.

SIZE

Level 1	→	Level 2	Level 1
"*string*"	→		*n*
{ *list* }	→		*n*
[*vector*]	→		{ *n* }
[[*matrix*]]	→		{ *n m* }
'*symb*'	→		*n*
grob	→	$\#n_{width}$	$\#m_{height}$
PICT	→	$\#n_{width}$	$\#m_{height}$
x_unit	→		*n*

Keyboard Access:

CHARS SIZE

PRG LIST ELEM SIZE

PRG GROB NXT SIZE

Affected by Flags: None

Remarks: The size of a unit is computed as follows: the scalar (+1), the underscore (+1), each unit name (+1), operator or exponent (+1), and each prefix (+2).

Any object type not listed above returns a value of 1.

Related Commands: CHR, NUM, POS, REPL, SUB

SL

Shift Left Command: Shifts a binary integer one bit to the left.

{ }

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Keyboard Access: (MTH) BASE (NXT) BIT SL

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The most significant bit is shifted out to the left and lost, while the least significant bit is regenerated as a zero. SL is equivalent to binary multiplication by 2, truncated to the current wordsize.

Related Commands: ASR, SLB, SR, SRB

SLB

Shift Left Byte Command: Shifts a binary integer one byte to the left.

{ }

Level 1	→	Level 1
#n_1	→	#n_2

Keyboard Access: (MTH) BASE (NXT) BYTE SLB

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The most significant byte is shifted out to the left and lost, while the least significant byte is regenerated as zero. SLB is equivalent to binary multiplication by 2^8 (or executing SL eight times), truncated to the current wordsize.

Related Commands: ASR, SL, SR, SRB

SLOPEFIELD

SLOPEFIELD Plot Type Command: Sets the plot type to SLOPEFIELD.

Keyboard Access: ◀ PLOT NXT 3D FTYPE SLOPE

Affected by Flags: None

Remarks: When plot type is set to SLOPEFIELD, the DRAW command plots a slope representation of a scalar function with two variables. SLOPEFIELD requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

{ x_{left} x_{right} y_{near} y_{far} z_{low} z_{high} x_{min} x_{max} y_{min} y_{max} x_{eye} y_{eye} z_{eye} x_{step} y_{step} }

For plot type SLOPEFIELD, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

For plot type SLOPEFIELD, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is not used.

- (x_{max}, y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is X.
- *res* is not used.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command SLOPEFIELD places the command name SLOPEFIELD in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is Y.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, TRUTH, WIREFRAME, YSLICE

SNEG

Store Negate Command: Replaces the contents of a variable with its negative.

{ }

Level 1	→	Level 1
'*name*'	→	

Keyboard Access: ◀ MEMORY MATH NXT SNEG

Affected by Flags: None

Remarks: The named object must be a number, an array, an algebraic object, a unit object, or a graphics object. For information on negation, see NEG.

Related Commands: NEG, SCONJ, SINV

SNRM

Spectral Norm Command: Returns the spectral norm of an array.

{ }

Level 1	→	Level 1
[*array*]	→	$x_{\text{spectralnorm}}$

Keyboard Access: (MTH) MATR NORM SNRM

Affected by Flags: None

Remarks: The spectral norm of a vector is its Euclidean length, and is equal to the largest singular value of a matrix.

Related Commands: ABS, CNRM, COND, RNRM, SRAD, TRACE

SOLVEQN

Start Equation Solver Command: Starts the appropriate solver for a specified set of equations.

{ }

Level 3	Level 2	Level 1	→	Level 1
n	*m*	*0/1*	→	

Keyboard Access: ↶ (EQ LIB) EQLIB SOLVE

Affected by Flags: Unit Type (60), Units Usage (61)

Remarks: SOLVEQN sets up and starts the appropriate solver for the specified set of equations, bypassing the Equation Library catalogs. It sets *EQ* (and *Mpar* if more than one equation is being solved), sets the unit options according to flags 60 and 61, and starts the appropriate solver.

SOLVEQN uses subject and title numbers (levels 3 and 2) and a "PICT" option (level 1), and returns nothing. Subject and title numbers are listed in chapter 4. If the "PICT" option is 0, *PICT* is not affected; otherwise, the equation picture is copied into *PICT*.

Related Commands: EQNLIB, MSOLVER

SORT

Ascending Order Sort Command: Sorts the elements in a list in ascending order.

Level 1	→	Level 1
{ *list* }$_1$	→	{ *list* }$_2$

Keyboard Access:

PRG LIST PROC NXT SORT

MTH LIST SORT

Affected by Flags: None

Remarks: The elements in the list can be real numbers, strings, lists, names, binary integers, or unit objects. However, all elements in the list must all be of the same type. Strings and names are sorted by character code number. Lists of lists are sorted by the first element in each list.

To sort in reverse order, use SORT REVLIST.

Related Commands: REVLIST

SPHERE

Spherical Mode Command: Sets Spherical coordinate mode.

Keyboard Access:

(MTH) VECTR (NXT) SPHER

(↰) (MODES) ANGL SPHER

Affected by Flags: None

Remarks: SPHERE sets flags −15 and −16, and displays the RAD annunciator.

In Spherical mode, vectors are displayed as polar components. Therefore, a 3D vector would appear as $[r \angle\theta \angle\phi]$.

Related Commands: CYLIN, RECT

SQ

Square Analytic Function: Returns the square of the argument.

{ }

Level 1	→	Level 1
z	→	z^2
x_unit	→	x^2 *_unit*^{2}
[[*matrix*]]	→	[[*matrix* × *matrix*]]
'*symb*'	→	'SQ(*symb*)'

Keyboard Access: (↰) (x^2)

Affected by Flags: Numerical Results (−3)

Remarks: The square of a complex argument (x, y) is the complex number $(x^2 - y^2, 2xy)$.

Matrix arguments must be square.

Related Commands: √, ^

SR

Shift Right Command: Shifts a binary integer one bit to the right.

{ }

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Keyboard Access: (MTH) BASE (NXT) BIT SR

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The least significant bit is shifted out to the right and lost, while the most significant bit is regenerated as a zero. SR is equivalent to binary division by 2.

Related Commands: ASR, SL, SLB, SRB

SRAD

Spectral Radius Command: Returns the spectral radius of a square matrix.

{ }

Level 1	→	Level 1
[[*matrix*]]$_{n \times n}$	→	$x_{\text{spectralradius}}$

Keyboard Access: (MTH) MATR NORM SRAD

Affected by Flags: None

Remarks: The spectral radius of a matrix is a measure of the size of the matrix, and is equal to the absolute value of the largest eigenvalue of the matrix.

SRAD

Related Commands: COND, SNRM, TRACE

SRB

Shift Right Byte Command: Shifts a binary integer one byte to the right.

{ }

Level 1	→	Level 1
#n_1	→	#n_2

Keyboard Access: MTH BASE NXT BYTE SRB

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The least significant byte is shifted out to the right and lost, while the most significant byte is regenerated as zero. SRB is equivalent to binary division by 2^8 (or executing SR eight times).

Related Commands: ASR, SL, SLB, SR

SRECV

Serial Receive Command: Reads up to *n* characters from the serial input buffer and returns them as a string, along with a digit indicating whether errors occurred.

{ }

Level 1	→	Level 2	Level 1
n	→	'*string*'	0/1

Keyboard Access: ← I/O NXT SERIA SRECV

Affected by Flags: I/O Device (−33)

Remarks: SRECV does not use Kermit protocol.

If n characters are not received within the time specified by STIME (default is 10 seconds), SRECV "times out", returning a 0 to level 1 and as many characters as were received to level 2.

If the level 2 output from BUFLEN is used as the input for SRECV, SRECV will not have to wait for more characters to be received. Instead, it returns the characters already in the input buffer.

If you want to accumulate bytes in the input buffer before executing SRECV, you must first open the port using OPENIO (if the port isn't already open).

SRECV can detect three types of error when reading the input buffer:

- Framing errors and UART overruns (both causing "Receive Error" in ERRM).
- Input-buffer overflows (causing "Receive Buffer Overflow" in ERRM).
- Parity errors (causing "Parity Error" in ERRM).

SRECV returns 0 if an error is detected when reading the input buffer, or 1 if no error is detected.

Parity errors do not stop data flow into the input buffer. However, if a parity error occurs, SRECV stops reading data after encountering a character with an error.

Framing, overrun, and overflow errors cause all subsequently received characters to be ignored until the error is cleared. SRECV does not detect and clear any of these types of errors until it tries to read the byte where the error occurred. Since these three errors cause the byte where the error occurred and all subsequent bytes to be ignored, the input buffer will be empty after all previously received good bytes have been read. Therefore, SRECV detects and clears these errors when it tries to read a byte from an empty input buffer.

Note that BUFLEN also clears the above-mentioned framing, overrun, and overflow errors. Therefore, SRECV cannot detect an input-buffer overflow after BUFLEN is executed, unless more characters were received after BUFLEN was executed (causing the input buffer to overflow again). SRECV also cannot detect framing and UART overrun errors cleared by BUFLEN. To find where the data error

SRECV

occurred, save the number of characters returned by BUFLEN (which gives the number of "good" characters received), because as soon as the error is cleared, new characters can enter the input buffer.

Example: If 10 good bytes were received followed by a framing error, then an SRECV command told to read 10 bytes would *not* indicate an error. Only when SRECV tries to read the byte that caused the framing error does it return a 0. Similarly, if the input buffer overflowed, SRECV would not indicate an error until it tried to read the first byte that was lost due to the overflow.

Related Commands: BUFLEN, CLOSEIO, OPENIO, SBRK, STIME, XMIT

SST

Execute Program Step Operation: Returns and executes the next step of a program. If the next step is a subroutine, executes the subroutine in a single step.

Keyboard Access: PRG NXT RUN SST

Affected by Flags: None

Remarks: SST is not programmable.

Related Commands: NEXT (operation), SST↓

SST↓

Execute Subroutine Step Operation: Returns and executes the next step of a program or subroutine. If the next step is a subroutine, returns and executes the first step of the subroutine.

Keyboard Access: PRG NXT RUN SST↓

Affected by Flags: None

Remarks: SST↓ is not programmable.

Related Commands: NEXT (operation), SST

START

START Definite Loop Structure Command: Begins START ... NEXT and START ... STEP definite loop structures.

	Level 2	Level 1	→	Level 1
START	x_{start}	x_{finish}	→	
NEXT			→	
STEP		$x_{increment}$	→	
STEP		$'symb_{increment}'$	→	

Keyboard Access: PRG BRCH START START

Affected by Flags: None

Remarks: *Definite loop structures* execute a command or sequence of commands a specified number of times.

- START ... NEXT executes a portion of a program a specified number of times. The syntax is this:

  x_{start} x_{finish} START *loop-clause* NEXT

  START takes two numbers (x_{start} and x_{finish}) from the stack and stores them as the starting and ending values for a loop counter. Then the loop clause is executed. NEXT increments the counter by 1 and tests to see if its value is less than or equal to x_{finish}. If so, the loop clause is executed again. Notice that the loop clause is always executed at least once.

- START ... STEP works just like START ... NEXT, except that it can use an increment value other than 1. The syntax is this:

  x_{start} x_{finish} START *loop-clause* $x_{increment}$ STEP

  START takes two numbers (x_{start} and x_{finish}) from the stack and stores them as the starting and ending values of the loop counter. Then the loop clause is executed. STEP takes $x_{increment}$ from the stack and increments the counter by that value. If the argument of STEP is an algebraic or a name, it is automatically evaluated to a number.

START

The increment value can be positive or negative:

- If positive, the loop is executed again when the counter is less than or equal to x_{finish}.
- If negative, the loop is executed when the counter is greater than or equal to x_{finish}.

Related Commands: FOR, NEXT, STEP

STD

Standard Mode Command: Sets the number display format to Standard mode.

Keyboard Access: ↩ MODES FMT STD

Affected by Flags: None

Remarks: Executing STD has the same effect as clearing flags −49 and −50.

Standard format (ANSI Minimal BASIC Standard X3J2) produces the following results when displaying or printing a number:

- Numbers that can be represented exactly as integers with 12 or fewer digits are displayed without a fraction mark or exponent. Zero is displayed as 0.
- Numbers that can be represented exactly with 12 or fewer digits, but not as integers, are displayed with a fraction mark but no exponent. Leading zeros to the left of the fraction mark and trailing zeros to the right of the fraction mark are omitted.
- All other numbers are displayed in scientific notation (see SCI) with both a fraction mark (with one number to the left) and an exponent (of one to three digits). There are no leading or trailing zeros.

In algebraic objects, integers less than 10^3 are always displayed in Standard mode.

Example: The following table provides examples of numbers displayed in Standard mode:

Number	Displayed As	Representable With 12 Digits?
10^{11}	100000000000	Yes (integer)
10^{12}	1.E12	No
10^{-11}	.000000000001	Yes
1.2×10^{-11}	1.23E-11	No
12.345	12.345	Yes

Related Commands: ENG, FIX, SCI

STEP

STEP Command: Defines the increment (step) value, and ends definite loop structure.

See the FOR and START command entries for syntax information.

Keyboard Access:

PRG BRCH FOR STEP

PRG BRCH START STEP

Remarks: See the FOR and START keyword entries for more information.

Related Commands: FOR, NEXT, START

STEQ

Store in EQ Command: Stores an object into the reserved variable *EQ* in the current directory.

{ }

Level 1	→	Level 1
obj	→	

Keyboard Access: This command must be typed in, but you can store an object in *EQ* with:

◀ PLOT ◀ EQ

◀ PLOT NXT 3D ◀ EQ

Affected by Flags: None

Related Commands: RCEQ

STIME

Serial Time-Out Command: Specifies the period that SRECV (serial reception) and XMIT (serial transmission) wait before timing out.

{ }

Level 1	→	Level 1
x_{seconds}	→	
0	→	

Keyboard Access: ◀ I/O NXT SERIAL STIME

Affected by Flags: None

Remarks: The value for x is interpreted as a positive value from 0 to 25.4 seconds. If no value is given, the default is 10 seconds. If x is 0,

there is no time-out; that is, the device waits indefinitely, which can drain the batteries.

STIME is not used for Kermit time-out.

Related Commands: BUFLEN, CLOSEIO, SBRK, SRECV, XMIT

STO

Store Command: Stores an object into a specified variable or object.

{ }

Level 2	Level 1	→	Level 1
obj	'*name*'	→	
grob	*PICT*	→	
obj	:n_{port} :$name_{backup}$	→	
obj	'*name*(*index*)'	→	
backup	n_{port}	→	
library	n_{port}	→	
library	:n_{port} :$n_{library}$	→	

Keyboard Access: STO

Affected by Flags: None

Remarks: Storing a graphics object into *PICT* makes it the current graphics object.

To create a backup object, store the *obj* into the desired backup location (identified as :n_{port}:$name_{backup}$). STO will not overwrite an existing backup object.

To store backup objects and library objects, specify a port number (0 through 33). Ports 1 and 2 must be configured as independent RAM, since backup and library objects can be stored only in independent RAM (see the entry for FREE).

To use a library object, the object must be in a port and it must be attached. A library object from an application card (ROM) is

automatically in a port (1 through 33), but a library object copied into RAM (such as through the PC Link) must be stored into a port using STO.

After storing a library object in a port, it must then be attached to its directory before it can be used. To make a stored library "attachable", turn the calculator off and then on. (See the entry for ATTACH.) This action (storing a library object, then turning the calculator off and on) also causes the calculator to perform a *system halt*, which clears the stack, the LAST stack, and all local variables, and returns the MATH menu to the display.

STO can also replace a single element of an array or list stored in a variable. Specify the variable in level 1 as '*name*(*index*)', which is a user function with *index* as the argument. The *index* can be *n* or *n,m*, where *n* specifies the row position in a vector or list, and *n,m* specifies the row-and-column position in a matrix.

Example: `'A+B+C+D' 'SUMAD' STO` stores the expression A+B+C+D in the variable *SUMAD*.

`5 'A(3)' STO` stores the integer 5 in the third element in a list or vector *A*.

`2 'A(3,5)' STO` stores the integer 2 in the element in the third row and fifth column of matrix *A*.

Related Commands: DEFINE, RCL, →

STOALARM

Store Alarm Command: Stores an alarm in the system alarm list and returns its alarm index number.

Level 1	→	Level 1
x_{time}	→	n_{index}
{ *date time* }	→	n_{index}
{ *date time* obj_{action} }	→	n_{index}
{ *date time* obj_{action} x_{repeat} }	→	n_{index}

Keyboard Access: ← TIME ALRM STOAL

Affected by Flags: Date Format (−42), Repeat Alarms Not Rescheduled (−43), Acknowledged Alarms Saved (−44)

Remarks: If the argument is a real number x_{time}, the alarm date will be the current system date by default.

If obj_{action} is a string, the alarm is an appointment alarm, and the string is the alarm message. If obj_{action} is any other object type, the alarm is a control alarm, and the object is executed when the alarm comes due.

x_{repeat} is the repeat interval for the alarm in clock ticks, where 8192 clock ticks equals 1 second.

n_{index} is a real integer identifying the alarm based on its chronological position in the system alarm list.

Example: With flag −42 clear, this command:

```
{ 11.06 15.2530 RUN 491520 } STOALARM
```

sets a repeating control alarm for November 6 of the currently specified year, at 3:25:30 PM. The alarm action is to execute variable *RUN*. The repeat interval is 491520 clock ticks (1 minute).

Related Commands: DELALARM, FINDALARM, RCLALARM

STOF

Store Flags Command: Sets the states of the system flags or the system and user flags.

Level 1	→	Level 1
$\#n_{system}$	→	
{ $\#n_{system}$ $\#n_{user}$ }	→	

Keyboard Access: ← MODES FLAG NXT STOF

Affected by Flags: Binary Integer Wordsize (−5 through −10)

STOF

The current wordsize must be 64 bits (the default wordsize) to store all flags. For example, executing STOF with a 32-bit binary integer stores only flags −1 through −32 and *clears* the other system flags.

Remarks: With argument #n_{system}, STOF sets the states of the system flags (−1 through −64) only. With argument { #n_{system} #n_{user} }, STOF sets the states of both the system and user flags.

A bit with value 1 sets the corresponding flag; a bit with value 0 clears the corresponding flag. The rightmost (least significant) bit of #n_{system} and #n_{user} correspond to the states of system flag −1 and user flag +1, respectively. If #n_{system} or #n_{user} contains fewer than 64 bits, the unspecified most significant bits are taken to have value 0.

STOF can preserve the states of flags before a program executes and changes the states. RCLF can then recall the flag's states after the program is executed.

Related Commands: RCLF

STOKEYS

Store Key Assignments Command: Defines multiple keys on the user keyboard by assigning objects to specified keys.

Level 1	→	Level 1
{ obj_1 x_{key1} ... obj_n x_{keyn} }	→	
{ S obj_1 x_{key1} ... obj_n x_{keyn} }	→	
'S'	→	

Keyboard Access: ↰ MODES KEYS STOK

Affected by Flags: User-Mode Lock (−61) and User Mode (−62) affect the status of the user keyboard.

Remarks: x_{key} is a real number of the form $rc.p$ specifying the key by its row number r, its column number c, and its plane (shift) p. (For a definition of plane, see the entry for ASN.)

The optional initial list parameter or argument S restores all keys without user assignments to their *standard* key assignments on the user keyboard. This is meaningful only when all standard key assignments had been suppressed (for the user keyboard) by the command 'S' DELKEYS (see DELKEYS).

If the argument *obj* is the name 'SKEY', the specified key is restored to its *standard key* assignment.

Related Commands: ASN, DELKEYS, RCLKEYS

STO+

Store Plus Command: Adds a number or other object to the contents of a specified variable.

{}

Level 2	Level 1	→	Level 1
obj	'*name*'	→	
'*name*'	*obj*	→	

Keyboard Access: ↰ MEMORY MATH STO+

Affected by Flags: None

Remarks: The object on the stack and the object in the variable must be suitable for addition to each other. STO+ can add any combination of objects suitable for stack addition (see the entry for +).

Using STO+ to add two arrays (where *obj* is an array and *name* is the global name of an array) requires less memory than using the stack to add them.

Related Commands: STO−, STO∗, STO/, +

STO−

Store Minus Command: Calculates the difference between a number (or other object) and the contents of a specified variable, and stores the new value to the specified variable.

{ }

Level 2	Level 1	→	Level 1
obj	'*name*'	→	
'*name*'	*obj*	→	

Keyboard Access: ◀ MEMORY MATH STO−

Affected by Flags: None

Remarks: The object on the stack and the object in the variable must be suitable for subtraction with each other. STO− can subtract any combination of objects suitable for stack subtraction (see the entry for −).

Using STO− to subtract two arrays (where *obj* is an array and *name* is the global name of an array) requires less memory than using the stack to subtract them.

Related Commands: STO+, STO∗, STO/, −

STO∗

Store Times Command: Multiplies the contents of a specified variable by a number or other object.

{ }

Level 2	Level 1	→	Level 1
obj	'*name*'	→	
'*name*'	*obj*	→	

Keyboard Access: ↩ MEMORY ARITH STO*

Affected by Flags: None

Remarks: The object on the stack and the object in the variable must be suitable for multiplication with each other. When multiplying two arrays, the result depends on the order of the arguments. The new object of the named variable is the level 2 array times the level 1 array. The arrays must be conformable for multiplication.

Using STO* to multiply two arrays or to multiply a number and an array (where *obj* is an array or a number and *name* is the global name of an array) requires less memory than using the stack to multiply them.

Related Commands: STO+, STO−, STO/, *

STO/

Store Divide Command: Calculates the quotient of a number (or other object) and the contents of a specified variable, and stores the new value to the specified variable.

{ }

Level 2	Level 1	→	Level 1
obj	'*name*'	→	
'*name*'	*obj*	→	

Keyboard Access: ↩ MEMORY ARITH STO/

Affected by Flags: None

Remarks: The new object of the specified variable is the level 2 object divided by the level 1 object.

The object on the stack and the object in the variable must be suitable for division with each other. If both objects are arrays, the divisor (level 1) must be a square matrix, and the dividend (level 2) must have the same number of columns as the divisor.

STO/

Using STO/ to divide one array by another array or to divide an array by a number (where *obj* is an array or a number and *name* is the global name of an array) requires less memory than using the stack to divide them.

Related Commands: STO+, STO−, STO*, /

STOΣ

Store Sigma Command: Stores *obj* in the reserved variable ΣDAT.

{ }

Level 1	→	Level 1
obj	→	

Keyboard Access: This command must be typed in, but you can store an object in ΣDAT with either of the following:

◀ PLOT NXT STAT ◀ ΣDAT

◀ STAT DATA ◀ ΣDAT

Affected by Flags: None

Remarks: STOΣ accepts any object and stores it in ΣDAT. However, if the object is not a matrix or the name of a variable containing a matrix, an `Invalid Σ Data` error occurs upon subsequent execution of a statistics command.

Related Commands: CLΣ, RCLΣ, Σ+, Σ−

STREAM

Stream Execution Command: Moves the first two elements from the list onto the stack, and executes *obj*. Then moves the next element (if any) onto the stack, and executes *obj* again using the previous result and the new element. Repeats this until the list is exhausted, and returns the final result.

Level 2	Level 1	→	Level 1
{ *list* }	*obj*	→	*result*

Keyboard Access: PRG LIST PROG STREAM

Affected by Flags: None

Remarks: STREAM is nominally designed for *obj* to be a program or command that requires two arguments and returns one result.

Examples: { 1 2 3 4 5 } «*» STREAM returns 120.

«+» STREAM is equivalent to ΣLIST.

Related Commands: DOSUBS

STR→

Evaluate String Command: Evaluates the text of a string as if the text were entered from the command line.

{ }

Level 1	→	Level 1
"*obj*"	→	*evaluated-object*

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: OBJ→ also includes this function. STR→ is included for compatibility with the HP 28S.

Related Commands: ARRY→, DTAG, EQ→, LIST→, OBJ→, →STR

→STR

Object to String Command: Converts any object to string form.

Level 1	→	Level 1
obj	→	"*obj*"

Keyboard Access:

◀ CHARS NXT →STR

PRG TYPE →STR

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12), Number Display Format (−49, −50)

Remarks: The full-precision internal form of a number is not necessarily represented in the result string. To ensure that →STR preserves the full precision of a number, select Standard number display format or a wordsize of 64 bits (or both) before executing →STR.

The result string includes the entire object, even if the displayed form of the object is too large to fit in the display.

If the argument object is normally displayed in two or more lines, the result string will contain newline characters (character 10) at the end of each line. The newlines are displayed as the character ▪.

If the argument object is already a string, →STR returns the string.

Example: →STR can create special displays to label program output or provide prompts for input. The sequence

```
"Result = " SWAP →STR + 1 DISP 1 FREEZE
```

displays `Result. = object` in line 1 of the display, where *object* is a string form of an object taken from level 1.

Related Commands: →ARRY, →LIST, STR→, →TAG, →UNIT

STWS

Set Wordsize Command: Sets the current binary integer wordsize to n bits, where n is a value from 1 through 64 (the default is 64).

{ }

Level 1	→	Level 1
n	→	
#*n*	→	

Keyboard Access: (MTH) BASE (NXT) STWS

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: Values of n less than 1 or greater than 64 are interpreted as 1 or 64, respectively.

If the wordsize is smaller than an integer entered in the command line, then the *most* significant bits are not displayed upon entry. The truncated bits are still present internally (unless they exceed 64), but they are not used for calculations and they are lost when a command uses this binary integer as an argument.

Results that exceed the given wordsize are also truncated to the wordsize.

Related Commands: BIN, DEC, HEX, OCT, RCWS

SUB

Subset Command: Returns the portion of a string or list defined by specified positions, or returns the rectangular portion of a graphics object or *PICT* defined by two corner pixel coordinates.

Level 3	Level 2	Level 1	→	Level 1
[[*matrix*]]$_1$	$n_{\text{startposition}}$	$n_{\text{endposition}}$	→	[[*matrix*]]$_2$
[[*matrix*]]$_1$	{ n_{row} n_{column} }	$n_{\text{endposition}}$	→	[[*matrix*]]$_2$
[[*matrix*]]$_1$	$n_{\text{startposition}}$	{ n_{row} n_{column} }	→	[[*matrix*]]$_2$
[[*matrix*]]$_1$	{ n_{row} n_{column} }	{ n_{row} n_{column} }	→	[[*matrix*]]$_2$
"*string*$_{\text{target}}$"	$n_{\text{startposition}}$	$n_{\text{endposition}}$	→	"*string*$_{\text{result}}$"
{ *list*$_{\text{target}}$ }	$n_{\text{startposition}}$	$n_{\text{endposition}}$	→	{ *list*$_{\text{result}}$ }
grob$_{\text{target}}$	{ $\#n_1$ $\#m_1$ }	{ $\#n_2$ $\#m_2$ }	→	*grob*$_{\text{result}}$
grob$_{\text{target}}$	(x_1, y_1)	(x_2, y_2)	→	*grob*$_{\text{result}}$
PICT	{ $\#n_1$ $\#m_1$ }	{ $\#n_2$ $\#m_2$ }	→	*grob*$_{\text{result}}$
PICT	(x_1, y_1)	(x_2, y_2)	→	*grob*$_{\text{result}}$

Keyboard Access:

◀ CHARS SUB

PRG LIST SUB

PRG GRUB SUB

MTH MATR MAKE NXT SUB

Affected by Flags: None

Remarks: If $n_{\text{end position}}$ is less than $n_{\text{start position}}$, SUB returns an empty string or list. Values of n less than 1 are treated as 1; values of n exceeding the length of the string or list are treated as that length.

For graphics objects, a user-unit coordinate less than the minimum user-unit coordinate of the graphics object is treated as that minimum. A pixel or user-unit coordinate greater than the maximum pixel or user-unit coordinate of the graphics object is treated as that maximum.

Examples: { A B C D E } 2 4 SUB returns { B C D }.

"ABCDE" 0 10 SUB returns "ABCDE".

PICT { # 10d # 20d } { # 20d # 40d } SUB returns
GRAPHIC 11 × 21.

Related Commands: CHR, GOR, GXOR, NUM, POS, REPL, SIZE

SVD

Singular Value Decomposition Command: Returns the singular value decomposition of an $m \times n$ matrix.

{ }

Level 1	→	Level 3	Level 2	Level 1
[[*matrix*]]$_A$	→	[[*matrix*]]$_U$	[[*matrix*]]$_V$	[*vector*]$_S$

Keyboard Access: (MTH) MATR FACTR SVD

Affected by Flags: None

Remarks: SVD decomposes A into 2 matrices and a vector. U is an $m \times m$ orthogonal matrix, V is an $n \times n$ orthogonal matrix, and S is a real vector, such that $A = U \times diag(S) \times V$. S has length MIN(m,n) and contains the singular values of A in nonincreasing order. The matrix *diag(S)* is an $m \times n$ diagonal matrix containing the singular values S.

The computed results should minimize (within computational precision):

$$\frac{|A - U \cdot \mathrm{diag}(S) \cdot V|}{\min(m,n) \cdot |A|}$$

where diag(S) denotes the $m \times n$ diagonal matrix containing the singular values S.

Related Commands: DIAG→, MIN, SVL

SVL

Singular Values Command: Returns the singular values of an $m \times n$ matrix.

{}

Level 1	→	Level 1
[[*matrix*]]	→	[*vector*]

Keyboard Access: (MTH) MATR FACTR (NXT) SVL

Affected by Flags: None

Remarks: SVL returns a real vector that contains the singular values of an $m \times n$ matrix in nonincreasing order. The vector has length MIN(m,n).

Related Commands: MIN, SVD

SWAP

Swap Objects Command: Interchanges the first two objects on the stack.

Level 2	Level 1	→	Level 2	Level 1
obj_1	obj_2	→	obj_2	obj_1

Keyboard Access: (←) (SWAP)

Affected by Flags: None

Related Commands: DUP, DUPN, DUP2, OVER, PICK, ROLL, ROLLD, ROT

SYSEVAL

Evaluate System Object Command: Evaluates unnamed operating system objects specified by their memory addresses.

{ }

Level 1	→	Level 1
#$n_{address}$	→	

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Using SYSEVAL with random addresses can corrupt memory.

Example: Display the version letter of an HP 48 by executing `#30794h SYSEVAL` . Version A, for example, would display `"HPHP48-A"` .

Related Commands: EVAL, LIBEVAL

%T

Percent of Total Function: Returns the percent of the level 2 argument that is represented by the level 1 argument.

%T

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	*100y/x*
x	'*symb*'	→	'%T(*x*, *symb*)'
'*symb*'	*x*	→	'%T(*symb*, *x*)'
'$symb_1$'	'$symb_2$'	→	'%T($symb_1$, $symb_2$)'
x_unit_1	y_unit_2	→	$100y_unit_2/x_unit_1$
x_unit	'*symb*'	→	'%T(*x_unit*, *symb*)'
'*symb*'	*x_unit*	→	'%T(*symb*, *x_unit*)'

Keyboard Access: MTH REAL %T

Affected by Flags: Numerical Results (−3)

Remarks: If both arguments are unit objects, the units must be consistent with each other.

The dimensions of a unit object are dropped from the result, *but units are part of the calculation.*

For more information on using temperature units with arithmetic functions, refer to the entry for +.

Example: 1_m 500_cm %T returns 500, because 500 cm represents 500% of 1 m.

100_K 50_K %T returns 50.

Related Commands: %, %CH

→TAG

Stack to Tag Command: Combines objects in levels 1 and 2 to create tagged (labeled) object.

Level 2	Level 1	→	Level 1
obj	"*tag*"	→	:*tag*:*obj*
obj	'*name*'	→	:*name*:*obj*
obj	*x*	→	:*x*:*obj*

Keyboard Access: PRG TYPE →TAG

Affected by Flags: None

Remarks: The "*tag*" argument is a string of fewer than 256 characters.

Related Commands: →ARRY, DTAG, →LIST, OBJ→, →STR, →UNIT

TAIL

Last Listed Elements Command: Returns all but the first element of a list or string.

Level 1	→	Level 1
{ obj_1 ... obj_n }	→	{ obj_2 ... obj_n }
"$string_1$"	→	"$string_2$"

Keyboard Access:

PRG LIST ELEM NXT TAIL

◀ CHARS NXT TAIL

TAIL

Affected by Flags: None

Example: "tall" TAIL returns "all".

Related Commands: HEAD

TAN

Tangent Analytic Function: Returns the tangent of the argument.

{}

Level 1	→	Level 1
z	→	tan *z*
'*symb*'	→	'TAN(*symb*)'
$x_unit_{angular}$	→	tan ($x_unit_{angular}$)

Keyboard Access: (TAN)

Affected by Flags: Numerical Results (−3), Angle Mode (−17, −18), Infinite Result Exception (−22)

Remarks: For real arguments, the current angle mode determines the number's interpretation as an angle, unless the angular units are specified.

For a real argument that is an odd-integer multiple of 90 in Degrees mode, an Infinite Result exception occurs. If flag −22 is set (no error), the sign of the result (MAXR) matches that of the argument.

For complex arguments,

$$\tan(x+iy) = \frac{(\sin x)(\cos x) + i(\sinh y)(\cosh y)}{\sinh^2 y + \cos^2 x}$$

If the argument for TAN is a unit object, then the specified angular unit overrides the angle mode to determine the result. Integration and differentiation, on the other hand, always observe the angle mode. Therefore, to correctly integrate or differentiate expressions containing TAN with a unit object, the angle mode must be set to Radians (since this is a "neutral" mode).

Related Commands: ATAN, COS, SIN

TANH

Hyperbolic Tangent Analytic Function: Returns the hyperbolic tangent of the argument.

{ }

Level 1	→	Level 1
z	→	tanh *z*
'*symb*'	→	'TANH(*symb*)'

Keyboard Access: MTH HYP TANH

Affected by Flags: Numerical Results (−3)

Remarks: For complex arguments,

$$\tanh(x+iy) = \frac{\sinh 2x + i\ \sin 2y}{\cosh 2x + \cos 2y}$$

Related Commands: ATANH, COSH, SINH

TAYLR

Taylor's Polynomial Command: Calculates the nth order Taylor's polynomial of '*symb*' in the variable *global*.

{ }

Level 3	Level 2	Level 1	→	Level 1
'*symb*'	'*global*'	n_{order}	→	'$symb_{Taylor}$'

Keyboard Access: ← SYMBOLIC TAYLR

TAYLR

Affected by Flags: None

Remarks: The polynomial is calculated at the point $global = 0$ (called a MacLaurin series).

TAYLR always returns a symbolic result, regardless of the state of the Numeric Results flag (−3).

Example: The command sequence `'1+SIN(X)^2' 'X' 5 TAYLR` returns `'1+X^2-8/4!*X^4'`.

Related Commands: ∂, $\int$, Σ

TDELTA

Temperature Delta Function: Calculates a temperature change.

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	x_{delta}
x_unit1	*y_unit2*	→	x_unit1_{delta}
x_unit	'*symb*'	→	'TDELTA(*x_unit*,*symb*)'
'*symb*'	*y_unit*	→	'TDELTA(*symb*,*y_unit*)'
'$symb_1$'	'$symb_2$'	→	'TDELTA($symb_1$,$symb_2$)'

Keyboard Access: ↩ EQ LIB UTILS NXT TDELT

Affected by Flags: Numerical Results (−3)

Remarks: TDELTA subtracts two points on a temperature scale, yielding a temperature *increment* (not an actual temperature). *x* or *x_unit1* is the final temperature, and *y* or *y_unit2* is the initial temperature. If unit objects are given, the increment is returned as a unit object with the same units as *x_unit1*. If real numbers are given, the increment is returned as a real number.

Related Commands: TINC

TEACH

Teaching Examples Function: Creates an EXAMPLES (EXAM) subdirectory in the HOME directory and loads HP 48 programming, graphing, and solver examples from ROM into it.

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Items stored in the EXAMPLES subdirectory are deleted when CLTEACH is executed.

Related Commands: CLTEACH

TEXT

Show Stack Display Command: Displays the stack display.

Keyboard Access: PRG NXT OUT TEXT

Affected by Flags: None

Remarks: TEXT switches from the graphics display to the stack display. TEXT does not update the stack display.

Example: The command sequence DRAW 5 WAIT TEXT selects the graphics display and plots the contents of the reserved variable EQ (or reserved variable ΣDAT). It subsequently waits for 5 seconds, and then switches back from the graphics display to the stack display.

Related Commands: PICTURE, PVIEW

THEN

THEN Command: Starts the true-clause in conditional or error-trapping structure.

See the IF and IFERR entries for syntax information.

Keyboard Access:

PRG NXT ERROR IFERR THEN

PRG BRCH CASE THEN

PRG BRCH IF THEN

Remarks: See the IF and IFERR entries for more information.

Related Commands: CASE, ELSE, END, IF, IFERR

TICKS

Ticks Command: Returns the system time as a binary integer, in units of 1/8192 second.

Level 1	→	Level 1
	→	$\#n_{time}$

Keyboard Access: ↰ TIME TICKS

Affected by Flags: None

Remarks: TICKS enables elapsed time computations.

Example: If the result from a previous invocation from TICKS is in level 1, then TICKS SWAP - 8192 / returns a real number whose value is the elapsed time in seconds between the two invocations.

Related Commands: TIME

TIME

Time Command: Returns the system time in the form HH.MMSSs.

Level 1	→	Level 1
	→	*time*

Keyboard Access: ↰ TIME TIME

Affected by Flags: None

Remarks: *time* has the form *HH.MMSSs*, where *HH* is hours, *MM* is minutes, *SS* is seconds, and *s* is zero or more digits (as many as allowed by the current display mode) representing fractional seconds. *time* is always returned in 24-hour format, regardless of the state of the Clock Format flag (−41).

Related Commands: DATE, TICKS, TSTR

→TIME

Set System Time Command: Sets the system time.

{ }

Level 1	→	Level 1
time	→	

Keyboard Access: ↰ TIME →TIME

Affected by Flags: None

Remarks: *time* must have the form *HH.MMSSs*, where *HH* is hours, *MM* is minutes, *SS* is seconds, and *s* is zero or more digits (as many as allowed by the current display mode) representing fractional seconds. *time* must use 24-hour format.

→TIME

Example: 13.3341 →TIME sets the system time to 1:33:41 PM.

Related Commands: CLKADJ, →DATE

TINC

Temperature Increment Command: Calculates a temperature increment.

{ }

Level 2	Level 1	→	Level 1
$x_{initial}$	y_{delta}	→	x_{final}
x_unit1	y_unit2_{delta}	→	x_unit1_{final}
x_unit	'*symb*'	→	'TINC(*x_unit*,*symb*)'
'*symb*'	y_unit_{delta}	→	'TINC(*symb*,y_unit_{delta})'
'$symb_1$'	'$symb_2$'	→	'TINC($symb_1$,$symb_2$')'

Keyboard Access: ← EQ LIB UTILS NXT TINC

Affected by Flags: Numerical Results (−3)

Remarks: TINC adds a temperature *increment* (not an actual temperature) to a point on a temperature scale. Use a negative increment to subtract the increment from the temperature. $x_{initial}$ or *x_unit1* is the initial temperature, and y_{delta} or y_unit2_{delta} is the temperature increment. The returned temperature is the resulting final temperature. If unit objects are given, the final temperature is returned as a unit object with the same units as *x_unit1*. If real numbers are given, the final temperature is returned as a real number.

Related Commands: TDELTA

TLINE

Toggle Line Command: For each pixel along the line in *PICT* defined by the specified coordinates, TLINE turns off every pixel that is on, and turns on every pixel that is off.

Level 2	Level 1	→	Level 1
$(x_1\ ,\ y_1)$	$(x_2\ ,\ y_2)$	→	
$\{\ \#n_1\ \ \#m_1\ \}$	$\{\ \#n_2\ \ \#m_2\ \}$	→	

Keyboard Access: PRG PICT TLINE

Affected by Flags: None

Example: The following program toggles on and off 10 times the pixels on the line defined by user-unit coordinates (1,1) and (9,9). Each state is maintained for .25 seconds.

```
«
 ERASE 0 10 XRNG 0 10 YNRG
 { # 0d # 0d } PVIEW
 «
  1 10 START
    (1,1) (9,9) TLINE
    .25 WAIT
   NEXT
  »
»
```

Related Commands: ARC, BOX, LINE

TMENU

Temporary Menu Command: Displays a built-in menu, library menu, or user-defined menu.

Level 1	→	Level 1
x_{menu}	→	
{ $list_{definition}$ }	→	
'$name_{definition}$'	→	

Keyboard Access: ◀ MODES MENU TMEN

Affected by Flags: None

Remarks: TMENU works just like MENU, except for user-defined menus (specified by a list or by the name of a variable that contains a list). Such menus are displayed like a custom menu and work like a custom menu, but are not stored in reserved variable *CST*. Thus, a menu defined and displayed by TMENU cannot be redisplayed by evaluating *CST*.

See the MENU entry for a list of the HP 48 built-in menus and the corresponding menu numbers (x_{menu}).

Examples: `7 TMENU` displays the first page of the MTH MATR menu.

`48.02 TMENU` displays the second page of the UNITS MASS menu.

`768 TMENU` displays the first page of commands in library 768.

`{ A 123 "ABC" } TMENU` displays the custom menu defined by the list argument.

`'MYMENU' TMENU` displays the custom menu defined by the name argument.

Related Commands: MENU, RCLMENU

TOT

Total Command: Computes the sum of each of the m columns of coordinate values in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{sum}
	→	$[\ x_{sum1}\ x_{sum2}\ \cdots\ x_{summ}\ ]$

Keyboard Access: ◀ STAT 1'WHR TOT

Affected by Flags: None

Remarks: The sums are returned as a vector of m real numbers, or as a single real number if $m = 1$.

Related Commands: MAXΣ, MINΣ, MEAN, PSDEV, PVAR, SDEV, VAR

TRACE

Matrix Trace Command: Returns the trace of a square matrix.

{ }

Level 1	→	Level 1
$[[\ matrix\ ]]_{n \times n}$	→	x_{trace}

Keyboard Access: MTH MATR NORM NXT TRACE

Affected by Flags: None

Remarks: The trace of a square matrix is the sum of its diagonal elements.

TRANSIO

I/O Translation Command: Specifies the character translation option. These translations affect only ASCII Kermit transfers and files printed to the serial port.

{ }

Level 1	→	Level 1
n_{option}	→	

Keyboard Access: ↩ I/O IOPAR TRAN

Affected by Flags: None

Remarks: Legal values for n are as follows:

n	Effect
0	No translation
1	Translate character 10 (line feed only) to/from characters 10 and 13 (line feed with carriage return, the Kermit protocol) (the default value)
2	Translate characters 128 through 159 (80 through 9F hexadecimal)
3	Translate all characters (128 through 255)

Related Commands: BAUD, CKSM, PARITY

TRN

Transpose Matrix Command: Returns the (conjugate) transpose of a matrix.

{}

Level 1	→	Level 1
[[*matrix*]]	→	[[*matrix*]]$_{\text{transpose}}$
'*name*'	→	

Keyboard Access: (MTH) MATR MAKE TRN

Affected by Flags: None

Remarks: TRN replaces an $n \times m$ matrix **A** with an $m \times n$ matrix $\mathbf{A}^T$, where:

$\mathbf{A}_{ij}{}^T = \mathbf{A}_{ji}$ for real matrices

$\mathbf{A}_{ij}{}^T = \text{CONJ}(\mathbf{A}_{ji})$ for complex matrices

If the matrix is specified by *name*, $\mathbf{A}^T$ replaces **A** in *name*.

Example: [[2 3 1][4 6 9]] TRN returns [[2 4][3 6][1 9]].

Related Commands: CONJ

TRNC

Truncate Function: Truncates an object to a specified number of decimal places or significant digits, or to fit the current display format.

TRNC

{ }

Level 2	Level 1	→	Level 1
z_1	$n_{truncate}$	→	z_2
z_1	'$symb_{truncate}$'	→	'TRNC(z_1,$symb_{truncate}$)'
'$symb_1$'	$n_{truncate}$	→	'TRNC($symb_1$,$n_{truncate}$)'
'$symb_1$'	'$symb_{truncate}$'	→	'TRNC($symb_1$,$symb_{truncate}$)'
[*array*]$_1$	$n_{truncate}$	→	[*array*]$_2$
x_unit	$n_{truncate}$	→	*y_unit*
x_unit	'$symb_{truncate}$'	→	'TRNC(*x_unit*,$symb_{truncate}$)'

Keyboard Access: MTH REAL NXT NXT TRNC

Affected by Flags: Numerical Results (−3)

Remarks: $n_{truncate}$ (or 'symb$_{truncate}$' if flag −3 is set) controls how the level 2 argument is truncated, as follows:

$n_{truncate}$	Effect on Level 2 Argument
0 through 11	truncated to n decimal places
−1 through −11	truncated to n significant digits
12	truncated to the current display format

For complex numbers and arrays, each real number element is truncated. For unit objects, the number part of the object is truncated.

Examples: (4.5792,8.1275) 2 TRNC returns (4.57,8.12).

[2.84907 3.96351 2.78453] −2 TRNC returns
[2.8 3.9 2.7].

Related Commands: RND

TRUTH

Truth Plot Type Command: Sets the plot type to TRUTH.

Keyboard Access: ◀ PLOT PTYPE TRUTH

Affected by Flags: None

Remarks: When the plot type is TRUTH, the DRAW command plots the current equation as a truth-valued function of two real variables. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

$$\{\ (x_{min}, y_{min})\ (x_{max}, y_{max})\ indep\ res\ axes\ ptype\ depend\ \}$$

For plot type TRUTH, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (-6.5,-3.1).
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a name specifying the independent variable on the horizontal axis, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the horizontal plotting range). The default value is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable on the *horizontal* axis, or a binary integer specifying that interval in pixels. The default value is #0, which specifies an interval of 1 pixel.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command TRUTH places the name TRUTH in *ptype*.
- *depend* is a name specifying the independent variable on the vertical axis, or a list containing such a name and two numbers specifying

the minimum and maximum values for the independent variable on the vertical axis (the vertical plotting range). The default value is *Y*.

The contents of *EQ* must be an expression or program, and cannot be an equation. It is evaluated for each pixel in the plot region. The minimum and maximum values of the independent variables (the plotting ranges) can be specified in *indep* and *depend*; otherwise, the values in $\{x_{min}, y_{min}\}$ and $\{x_{max}, y_{max}\}$ (the display range) are used. The result of each evaluation must be a real number. If the result is zero, the state of the pixel is unchanged. If the result is nonzero, the pixel is turned on (made dark).

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, WIREFRAME, YSLICE

TSTR

Date and Time String Command: Returns a string derived from the date and time.

{}

Level 2	Level 1	→	Level 1
date	*time*	→	"*DOW DATE TIME*"

Keyboard Access: ◀ TIME NXT NXT TSTR

Affected by Flags: Date Format (−42), Time Format (−41)

Remarks: The string has the form "*DOW DATE TIME*", where *DOW* is a three-letter abbreviation of the day of the week corresponding to the argument *date* and *time*, *DATE* is the argument *date* in the current date format, and *TIME* is the argument *time* in the current time format.

Example: With flags −42 and −41 clear, `2.061990 14.55 TSTR` returns `"TUE 02/06/90 02:55:00P"`.

Related Commands: DATE, TICKS, TIME

TVARS

Typed Variables Command: Lists all global variables in the current directory that contain objects of the specified types.

Level 1	→	Level 1
n_{type}	→	{ *global* ... }
{ n_{type} ... }	→	{ *global* ... }

Keyboard Access: ◀ MEMORY DIR TVARS

Affected by Flags: None

Remarks: If the current directory contains no variables of the specified types, TVARS returns an empty list.

For a table of the object-type numbers, see the entry for TYPE.

Related Commands: PVARS, TYPE, VARS

TVM

TVM Menu Command: Displays the TVM Solver menu.

Keyboard Access: This command must be typed in, but you can also access the menu using ◀ SOLVE TVM SOLVR.

Affected by Flags: None

Related Commands: AMORT, TVMBEG, TVMEND, TVMROOT

TVMBEG

Payment at Start of Period Command: Specifies that TVM calculations treat payments as being made at the begining of the compounding periods.

Keyboard Access: This command must be typed in, but you can control begin/end mode with ◀ SOLVE TVM BEG.

Affected by Flags: None

Related Commands: AMORT, TVM, TVMEND, TVMROOT

TVMEND

Payment at End of Period Command: Specifies that TVM calculations treat payments as being made at the end of the compounding periods.

Keyboard Access: This command must be typed in, but you can control begin/end mode with ◀ SOLVE TVM BEG.

Affected by Flags: None

Related Commands: AMORT, TVM, TVMBEG, TVMROOT

TVMROOT

TVM Root Command: Solves for the specified TVM variable using values from the remaining TVM variables.

{ }

Level 1	→	Level 1
'*TVM variable*'	→	$x_{TVM\ variable}$

Keyboard Access: ◀ SOLVE TVM TVMR

Affected by Flags: None

Related Commands: AMORT, TVM, TVMBEG, TVMEND

TYPE

Type Command: Returns the type number of an object.

Level 1	→	Level 1
obj	→	n_{type}

Keyboard Access:

PRG TYPE NXT NXT TYPE

PRG TEST NXT TYPE

Affected by Flags: None

Remarks: The following table lists object types and their type numbers.

Object Type Numbers

Object Type	Number
User Objects:	
Real number	0
Complex number	1
Character string	2
Real array	3
Complex array	4
List	5
Global name	6
Local name	7
Program	8
Algebraic object	9
Binary integer	10

Object Type Numbers (continued)

Object Type	Number
Graphics object	11
Tagged object	12
Unit object	13
XLIB name	14
Directory	15
Library	16
Backup object	17
Built-in Commands:	
Built-in function	18
Built-in command	19
System Objects:	
System binary	20
Extended real	21
Extended complex	22
Linked array	23
Character	24
Code object	25
Library data	26
External object	27-31

The HP 28S TYPE command returns number 8 for built-in functions and built-in commands (HP 48 TYPE numbers 18 and 19).

Related Commands: SAME, TVARS, VTYPE, ==

UBASE

Convert to SI Base Units Function: Converts a unit object to SI base units.

{ }

Level 1	→	Level 1
x_unit	→	*y_base-units*
'*symb*'	→	'UBASE(*symb*)'

Keyboard Access: ◀ UNITS UBASE

Affected by Flags: Numerical Results (−3)

Example: 30_knot UBASE returns 15.433333333_m/s.

Related Commands: CONVERT, UFACT, →UNIT, UVAL

UFACT

Factor Unit Command: Factors the level 1 unit from the unit expression of the level 2 unit object.

{ }

Level 2	Level 1	→	Level 1
$x_1_unit_1$	$x_2_unit_2$	→	$x_3_unit_2 * unit_3$

Keyboard Access: ◀ UNITS UFACT

Affected by Flags: None

Remarks: UFACT is equivalent to this sequence:

```
OBJ→ 3 ROLLD / OVER / UBASE *
```

Example: 1_W 1_N UFACT returns 1_N*m/s.

Related Commands: CONVERT, UBASE, →UNIT, UVAL

→UNIT

Stack to Unit Object Command: Creates a unit object from a real number and the unit part of a unit object.

{}

Level 2	Level 1	→	Level 1
x	*y_unit*	→	*x_unit*

Keyboard Access:

PRG TYPE →UNIT

◀ UNITS →UNIT

Affected by Flags: None

Remarks: →UNIT adds units to a real number, combining the number and the unit part of a unit object (the numerical part of the unit object is ignored). →UNIT is the reverse of OBJ→ applied to a unit object.

Related Commands: →ARRY, →LIST, →STR, →TAG

UNTIL

UNTIL Command: Starts test-clause in a DO ... UNTIL ... END indefinite loop structure.

See the DO entry for syntax information.

Keyboard Access: PRG BRCH DO UNTIL

Remarks: See the DO entry for more information.

Related Commands DO, END

UPDIR

Up Directory Command: Makes the parent of the current directory the new current directory.

Keyboard Access: ◀ UP

Affected by Flags: None

Remarks: UPDIR has no effect if the current directory is *HOME*.

Related Commands: CRDIR, HOME, PATH, PGDIR

UTPC

Upper Chi-Square Distribution Command: Returns the probability utpc(n, x) that a chi-square random variable is greater than x, where n is the number of degrees of freedom of the distribution.

{ }

Level 2	Level 1	→	Level 1
n	x	→	$utpc(n, x)$

Keyboard Access: MTH NXT FFDE NXT UTPC

Affected by Flags: None

Remarks: The defining equations are these:

- For x $\geq$ 0:

$$utpc(n,x) = \left[\frac{1}{2^{\frac{n}{2}}\Gamma\left(\frac{n}{2}\right)}\right]\int_x^\infty t^{\frac{n}{2}-1}e^{-\frac{t}{2}}dt$$

- For x $<$ 0:

$$utpc(n,x) = 1$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the HP 48 factorial command.

UTPC

The value n is rounded to the nearest integer and, when rounded, must be positive.

Related Commands: UTPF, UTPN, UTPT

UTPF

Upper Snedecor's F Distribution Command: Returns the probability utpf(n_1, n_2, x) that a Snedecor's F random variable is greater than x, where n_1 and n_2 are the numerator and denominator degrees of freedom of the F distribution.

{}

Level 3	Level 2	Level 1	→	Level 1
n_1	n_2	x	→	$utpf(n_1, n_2, x)$

Keyboard Access: MTH NXT PROB NXT UTPF

Affected by Flags: None

Remarks: The defining equations for utpf(n_1, n_2, x) are these:

- For x $\geq$ 0:

$$\left(\frac{n_1}{n_2}\right)^{\frac{n_1}{2}} \left[\frac{\Gamma\left(\frac{n_1+n_2}{2}\right)}{\Gamma\left(\frac{n_1}{2}\right)\Gamma\left(\frac{n_2}{2}\right)}\right] \int_x^\infty t^{\frac{(n_1-2)}{2}} \left[1+\left(\frac{n_1}{n_2}\right)t\right]^{-\frac{(n_1+n_2)}{2}} dt$$

- For x $<$ 0:

$$utpf(n_1, n_2, x) = 1$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the HP 48 factorial command.

The values n_1 and n_2 are rounded to the nearest integers and, when rounded, must be positive.

Related Commands: UTPC, UTPN, UTPT

UTPN

Upper Normal Distribution Command: Returns the probability utpn(m, v, x) that a normal random variable is greater than x, where m and v are the mean and variance, respectively, of the normal distribution.

{ }

Level 3	Level 2	Level 1	→	Level 1
m	*v*	*x*	→	*utpn(m, v, x)*

Keyboard Access: MTH NXT PROB NXT UTPN

Affected by Flags: None

Remarks: For all x and m, and for $v > 0$, the defining equation is this:

$$utpn(m, v, x) = \left[\frac{1}{\sqrt{2\pi v}}\right] \int_x^\infty e^{-\frac{(t-m)^2}{2v}} dt$$

For $v = 0$, UTPN returns 0 for $x \geq m$, and 1 for $x < m$.

Related Commands: UTPC, UTPF, UTPT

UTPT

Upper Student's t Distribution Command: Returns the probability utpt(n, x) that a Student's t random variable is greater than x, where n is the number of degrees of freedom of the distribution.

{ }

Level 2	Level 1	→	Level 1
n	*x*	→	*utpt(n, x)*

Keyboard Access: MTH NXT PROB NXT UTPT

UTPT

Affected by Flags: None

Remarks: The following is the defining equation for all x.

$$utpt(n,x) = \left[\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\sqrt{n\pi}}\right]\int_x^\infty \left(1+\frac{t^2}{n}\right)^{-\frac{n+1}{2}} dt$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the HP 48 factorial command.

The value n is rounded to the nearest integer and, when rounded, must be positive.

Related Commands: UTPC, UTPF, UTPN

UVAL

Unit Value Function: Returns the numerical part of a unit object.

UVAL *Unit Value* **Function**

Level 1	→	Level 1
x_unit	→	*x*
'*symb*'	→	'UVAL(*symb*)'

Keyboard Access: ◀ UNITS UVAL

Affected by Flags: Numerical Results (−3)

Related Commands: CONVERT, UBASE, UFACT, →UNIT

VAR

Variance Command: Calculates the sample variance of the coordinate values in each of the m columns in the current statistics matrix (ΣDAT).

Level 1	→	Level 1
	→	$x_{variance}$
	→	[$x_{variance1}$ ⋯ $x_{variancem}$]

Keyboard Access: ◀ STAT 1.VAR NXT 2.VAR

Affected by Flags: None

Remarks: The variance (equal to the square of the standard deviation) is returned as a vector of m real numbers, or as a single real number if $m = 1$. The variances are computed using this formula:

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

where x_i is the ith coordinate value in a column, $\overline{x}$ is the mean of the data in this column, and n is the number of data points.

Related Commands: MAXΣ, MEAN, MINΣ, PSDEV, PVAR, SDEV, TOT

VARS

Variables Command: Returns a list of all variables' names in the VAR menu (the current directory).

Level 1	→	Level 1
	→	{ $global_1$ … $global_n$ }

VARS

Keyboard Access: ◀ MEMORY DIR VARS

Affected by Flags: None

Related Commands: ORDER, PVARS, TVARS

VERSION

Software Version Command: Displays the software version and copyright message.

Level 1	→	Level 2	Level 1
	→	"*version number*"	"*copyright message*"

Keyboard Access: None. Must be typed in.

Affected by Flags: None

VTYPE

Variable Type Command: Returns the type number of the object contained in the named variable.

{ }

Level 1	→	Level 1
'*name*'	→	n_{type}
$:n_{port} : name_{backup}$	→	n_{type}
$:n_{port} : n_{library}$	→	n_{type}

Keyboard Access: PRG TYPE NXT NXT VTYPE

Affected by Flags: None

Remarks: If the named variable does not exist, VTYPE returns −1.

For a table of the objects' type numbers, see the entry for TYPE.

Related Commands: TYPE

→V2

Stack to Vector/Complex Number Command: Converts two numbers from the stack into a 2-element vector or a complex number.

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	[*x y*]
x	*y*	→	[*x* ∠*y*]
x	*y*	→	(*x*, *y*)
x	*y*	→	(*x*, ∠*y*)

Keyboard Access: (MTH) VECTR →V2

Affected by Flags: Complex Mode (−19), Coordinate System (−16)

Remarks: The result returned depends on the setting of flags −16 and and −19, as shown in the following table:

	Flag −19 clear	**Flag −19 set**
Flag −16 clear (Rectangular mode)	[x y]	(x, y)
Flag −16 set (Polar mode)	[x ∠y]	(x, ∠y)

Examples: With flag −19 clear, and flags −16 clear, 2 3 →V2 returns [2 3].

With flag −19 set and flag −16 set (Polar/Spherical mode), 2 3 →V2 returns (2,∠3).

→V2

Related Commands: V→, →V3

→V3

Stack to 3-Element Vector Command: Converts three numbers into a 3-element vector.

{ }

Level 3	Level 2	Level 1	→	Level 1
x_1	x_2	x_3	→	[x_1 x_2 x_3]
x_1	x_{theta}	x_z	→	[x_1 ∠x_{theta} x_z]
x_1	x_{theta}	x_{phi}	→	[x_1 ∠x_{theta} ∠x_{phi}]

Keyboard Access: (MTH) VECTR →V3

Affected by Flags: Coordinate System (−15 and −16)

Remarks: The result returned depends on the coordinate mode used, as shown in the following table:

Mode	Result
Rectangular (flag −16 clear)	[x_1 x_2 x_3]
Polar/Cylindrical (flag −15 clear and −16 set)	[x_1 x∠$_{theta}$ x_z]
Polar/Spherical (flag −15 and −16 set)	[x_1 x∠$_{theta}$ x∠$_{phi}$]

Examples: With flag −16 clear (Rectangular mode), 1 2 3 →V3 returns [1 2 3].

With flag −15 clear and −16 set (Polar/Cylindrical mode), 1 2 3 →V3 returns [1 ∠2 3].

With flags −15 and −16 set (Polar/Spherical mode), `1 2 3 →V3` returns `[ 1 ∠2 ∠3 ]`.

Related Commands: V→, →V2

V→

Vector/Complex Number to Stack Command: Separates a vector or complex number into its component elements.

{}

Level 1	→	Level n .. Level 3	Level 2	Level 1
[*x y*]	→		*x*	*y*
[x_r ∠y_{theta}]	→		x_r	y_{theta}
[x_1 x_2 x_3]	→	x_1	x_2	x_3
[x_1 ∠x_{theta} x_z]	→	x_1	x_{theta}	x_z
[x_1 ∠x_{theta} ∠x_{phi}]	→	x_1	x_{theta}	x_{phi}
[x_1 x_2 … x_n]	→	x_1 … x_{n-2}	x_{n-1}	x_n
(*x*, *y*)	→		*x*	*y*
(x_r, ∠y_{theta})	→		x_r	y_{theta}

Keyboard Access: (MTH) VECTR →V→

Affected by Flags: Coordinate System (−15 and −16)

The elements of the argument complex number or vector are converted from their values in Rectangular mode (the form in which the complex number or vector is stored internally) to the current coordinate system mode before being returned to the stack. This means that the element values returned to the stack always match the *displayed* element values of the argument vector or complex number.

Remarks: For vectors with four or more elements, V→ executes *independently* of the coordinate system mode, and always returns the elements of the vector to the stack as they are stored internally (in rectangular form). Thus, V→ is equivalent to OBJ→ for vectors with four or more elements.

V→

Examples: With flag −16 clear (Rectangular mode), `(2,3) V→` returns `2` to level 2 and `3` to level 1.

With flag −15 clear and flag −16 set (Polar/Cylindrical mode), `[ 2 ∠7 4 ] V→` returns `2` to level 3, `7` to level 2, and `4` to level 1.

`[ 9 7 5 3 ] V→` returns `9` to level 4, `7` to level 3, `5` to level 2, and `3` to level 1, independent of the state of flags −15 and −16.

Related Commands: →V2, →V3

---

*W

Multiply Width Command: Multiplies a plot's horizontal scale by x_{factor}.

{ }

Level 1	→	Level 1
x_{factor}	→	

Keyboard Access: ◀ PLOT PPAR NXT +|-|

Affected by Flags: None

Remarks: Executing *W changes the x-axis display range (x_{min} and x_{max} in the reserved variable *PPAR*). The plot center (the user-unit coordinate of the center pixel) does not change.

Related Commands: AUTO, *H, XRNG

WAIT

Wait Command: Suspends program execution for specified time, or until a key is pressed.

{}

Level 1	→	Level 1
x	→	
0	→	x_{key}
−1	→	x_{key}

Keyboard Access: PRG NXT IN WAIT

Affected by Flags: None

Remarks: The function of WAIT depends on the argument, as follows:

- Argument x interrupts program execution for x seconds.
- Argument 0 suspends program execution until a valid key is pressed (see below). WAIT then returns x_{key}, which defines where the pressed key is on the keyboard, and resumes program execution.

  x_{key} is a three-digit number that identifies a key's location on the keyboard. See the entry for ASN for a description of the format of x_{key}.
- Argument −1 works as with argument 0, except that the currently specified menu is also displayed.

◀, ▶, α, α◀, and α▶ are not by themselves valid keys.

Arguments 0 or -1 do not affect the display, so that messages persist even though the keyboard is ready for input (FREEZE is not required).

Normally, the MENU command does not update the menu keys until a program halts or ends. WAIT with argument −1 enables a previous execution of MENU to display that menu while the program is suspended for a key press.

WAIT

Examples: This program:

```
« "Press [1] to add■Press any other key to subtract"
1 DISP 0 WAIT IF 82.1 SAME THEN + ELSE − END »
```

displays a prompting message and halts program execution until a key is pressed. If the 1 key (location 82.1) is pressed, two numbers on the stack are added. If any other key is pressed, two numbers on the stack are subtracted.

This program:

```
« { ADD { } { } { } { } SUB } MENU "Press [ADD]
to add■ Press [SUB] to subtract" 1 DISP −1 WAIT
IF 11.1 SAME THEN + ELSE − END »
```

builds a custom menu with labels ADD and SUB and a prompting message. Executing -1 WAIT displays the custom menu (note that it's not active) and suspends execution for keyboard input. If the ADD menu key (location 11.1) is pressed, two numbers on the stack are added. If any other key is pressed, two numbers on the stack are subtracted.

Related Commands: KEY

WHILE

WHILE Indefinite Loop Structure Command: Starts the WHILE ... REPEAT ... END indefinite loop structure.

	Level 1	→	Level 1
WHILE		→	
REPEAT	*T/F*	→	
END		→	

Keyboard Access: PRG BRCH WHILE WHILE

Affected by Flags: None

Remarks: WHILE ... REPEAT ... END repeatedly evaluates a test and executes a loop clause if the test is true. Since the test clause occurs before the loop-clause, the loop clause is never executed if the test is initially false. The syntax is this:

WHILE *test-clause* REPEAT *loop-clause* END

The test clause is executed and must return a test result to the stack. REPEAT takes the value from the stack. If the value is not zero, execution continues with the loop clause; otherwise, execution resumes following END.

Related Commands: DO, END, REPEAT

WIREFRAME

WIREFRAME Plot Type Command: Sets the plot type to WIREFRAME.

Keyboard Access: ◀ PLOT NXT 3D FTYPE WIREF

Affected by Flags: None

Remarks: When the plot type is set to WIREFRAME, the DRAW command plots a perspective view of the graph of a scalar function of two variables. WIREFRAME requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

{ x_{left} x_{right} y_{near} y_{far} z_{low} z_{high} x_{min} x_{max} y_{min} y_{max} x_{eye} y_{eye} z_{eye} x_{step} y_{step} }

For plot type WIREFRAME, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.

WIREFRAME

- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

For plot type WIREFRAME, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is not used.
- (x_{max}, y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is X.
- *res* is not used.
- *axes* is not used.
- *ptype* is a name specifying the plot type. Executing the command WIREFRAME places the command name WIREFRAME in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is Y.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, YSLICE

WSLOG

Warmstart Log Command: Returns four strings recording the date, time, and cause of the four most recent warmstart events.

Level 1	→	Level 4 ... Level 1
	→	"log_4" ... "log_1"

Keyboard Access: None. Must be typed in.

Affected by Flags: Date Format (−42)

Remarks: Each string "log_n" has the form "*code*−*date time*". The following table summarizes the legal values of *code* and their meanings.

Code	Description
0	The warmstart log was cleared by pressing ON SPC and then ON to wake the calculator up. ON SPC puts the HP 48 in "Coma mode" (very low power *with the system clock stopped*). Pressing ON then clears the log and warmstarts the system.
1	The interrupt system detected a very low battery condition at the battery contacts (not the same as a low system voltage), and put the calculator in a "Deep Sleep mode" (*with the system clock running*). When ON is pressed after the battery voltage is restored, the system warmstarts and puts a 1 in the log.
2	Hardware failed during IR transmission (timeout).
3	Run through address 0.
4	System time is corrupt.
5	A Deep Sleep wakeup (for example, ON, Alarm) detected no change to port status, but some changes in data on one or both cards.

Code	Description
6	Unused.
7	A 5-nibble word (CMOS test word) in RAM was corrupt. (This word is checked on every interrupt, but it is used only as an indicator of potentially corrupt RAM.)
8	One of the following anomalies involving device configuration was detected: ■ The interrupt system detected that one of the five devices was not configured. ■ During a warmstart, an unexpected device ID chain was encountered while attempting to configure 3 (Port1, Port2, Xtra) of the 5 devices. ■ Same as previous, but detected during Deep Sleep wakeup.
9	The alarm list is corrupt.
A	Unused.
B	The card module was removed (or card bounce).
C	Hardware reset occurred (for example, an electrostatic-discharge or user reset).
D	An expected System (RPL) error handler was not found in runstream.
E	The configuration table is corrupt (bad checksum for table data).
F	The system RAM card was removed.

The date and time stamp (*date time*) part of the log may be displayed as 00 ... 0000 for one of three reasons:

- The system time was corrupt when the stamp was recorded.
- The date and time stamp itself is corrupt (bad checksum).
- Fewer than four warmstarts have occurred since the log was last cleared.

ΣX

Sum of x-Values Command: Sums the values in the independent-variable column of the current statistical matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{sum}

Keyboard Access: ◀ STAT SUMS ΣX

Affected by Flags: None

Remarks: The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable ΣPAR. The default independent-variable column number is 1.

Related Commands: NΣ, XCOL, ΣX*Y, ΣX^2, ΣY, ΣY^2

ΣX^2

Sum of Squares of x-Values Command: Sums the squares of the values in the independent-variable column of the current statistical matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{sum}

Keyboard Access: ◀ STAT SUMS ΣX^2

Affected by Flags: None

Remarks: The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable ΣPAR. The default independent-variable column number is 1.

ΣX^2

Related Commands: NΣ, ΣX, XCOL, ΣX*Y, ΣY, ΣY^2

XCOL

Independent Column Command: Specifies the independent-variable column of the current statistics matrix (reserved variable ΣDAT).

{ }

Level 1	→	Level 1
n_{col}	→	

Keyboard Access: ↰ STAT ΣPAR XCOL

Affected by Flags: None

Remarks: The independent-variable column number is stored as the first parameter in the reserved variable ΣPAR. The default independent-variable column number is 1.

XCOL will accept a noninteger real number and store it in ΣPAR, but subsequent commands that utilize the XCOL specification in ΣPAR will cause an error.

Related Commands: BARPLOT, BESTFIT, COLΣ, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, YCOL

XMIT

Serial Transmit Command: Sends a string serially without using Kermit protocol, and returns a single digit that indicates whether the transmission was successful.

{}

Level 1	→	Level 2	Level 1
"*string*"	→		1
"*string*"	→	"$substring_{unsent}$"	0

Keyboard Access: ◀ I/O NXT SERIA XMIT

Affected by Flags: I/O Device (−33)

Remarks: XMIT is useful for communicating with non-Kermit devices such as RS-232 printers.

If the transmission is successful, XMIT returns a 1. If the transmission is not successful, XMIT returns the unsent portion of the string and a 0. Use ERRM to get the error message.

After receiving an XOFF command (with *transmit pacing* in the reserved variable *IOPAR* set), XMIT stops transmitting and waits for an XON command. XMIT resumes transmitting if an XON is received before the time-out set by STIME elapses; otherwise, XMIT terminates, returns a 0, and stores "Timeout" in ERRM.

Related Commands: BUFLEN, SBRK, SRECV, STIME

XOR

Exclusive OR Function: Returns the logical exclusive OR of two arguments.

{ }

Level 2	Level 1	→	Level 1
#n_1	#n_2	→	#n_3
"$string_1$"	"$string_2$"	→	"$string_3$"
T/F_1	T/F_2	→	0/1
T/F	'$symb$'	→	'T/F XOR $symb$'
'$symb$'	T/F	→	'$symb$ XOR T/F'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ XOR $symb_2$'

Keyboard Access:

MTH BASE NXT LOGIC XOR

PRG TEST NXT XOR

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: When the arguments are binary integers or strings, XOR does a bit-by-bit (base 2) logical comparison:

- Binary integer arguments are treated as sequences of bits with length equal to the current wordsize. Each bit in the result is determined by comparing the corresponding bits (bit_1 and bit_2) in the two arguments, as shown in the following table.

bit_1	bit_2	bit_1 **XOR** bit_2
0	0	0
0	1	1
1	0	1
1	1	0

- String arguments are treated as sequences of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must be the same length.

When the arguments are real numbers or symbolics, XOR simply does a true/false test. The result is 1 (true) if either, but not both, arguments are nonzero; it is 0 (false) if both arguments are nonzero or zero. This test is usually done to compare two test results.

If either or both of the arguments are algebraic objects, then the result is an algebraic of the form '$symb_1$ XOR $symb_2$'. Execute →NUM (or set flag −3 before executing XOR) to produce a numeric result from the algebraic result.

Related Commands: AND, NOT, OR

XPON

Exponent Function: Returns the exponent of the argument.

{}

Level 1	→	Level 1
x	→	n_{expon}
'$symb$'	→	'XPON($symb$)'

Keyboard Access: MTH REAL NXT XPON

Affected by Flags: Numerical Results (−3)

Examples: 1.2E34 XPON returns 34.

12.4E3 XPON returns 4.

'A*1E34 XPON returns 'XPON(A*1E34)'.

Related Commands: MANT, SIGN

XRECV

XModem Receive Command: Prepares the HP 48 to receive an object via XModem. The received object is stored in the given variable name.

Level 1	→	Level 1
'*name*'	→	

Keyboard Access: ◀ I/O NXT XRECV

Affected by Flags: I/O Device (−33), RECV Overwrite (−36)

Remarks: The transfer will start more quickly if you start the XModem sender *before* executing XRECV.

Invalid object names cause an error. If flag −36 is clear, object names that are already in use also cause an error.

If you are transferring data between two HP 48s, executing « AAA BBB CCC » XRECV receives *AAA*, *BBB*, and *CCC*. You also need to use a list on the sending end (« AAA BBB CCC » XSEND, for example).

Related Commands: BAUD, RECV, RECN, SEND, XSEND

XRNG

x-Axis Display Range Command: Specifies the x-axis display range.

{ }

Level 2	Level 1	→	Level 1
x_{min}	x_{max}	→	

Keyboard Access: ◀ PLOT PPAR XRNG

Affected by Flags: None

Remarks: The x-axis display range is stored in the reserved variable $PPAR$ as x_{min} and x_{max} in the complex numbers (x_{min}, y_{min}) and (x_{max}, y_{max}). These complex numbers are the first two elements of $PPAR$ and specify the coordinates of the lower left and upper right corners of the display ranges.

The default values of x_{min} and x_{max} are -6.5 and 6.5, respectively.

Related Commands: AUTO, PDIM, PMAX, PMIN, YRNG

XROOT

xth Root of y Command: Computes the xth root of a real number.

{ }

Level 2	Level 1	→	Level 1
y	x	→	$\sqrt[x]{y}$
'$symb_1$'	'$symb_2$'	→	'XROOT($symb_2$, $symb_1$)'
'$symb$'	x	→	'XROOT(x, $symb$)'
y	'$symb$'	→	'XROOT($symb$, y)'
y_unit	x	→	$\sqrt[x]{y}_unit^{1/x}$
y_unit	'$symb$'	→	'XROOT($symb$, y_unit)'

Keyboard Access: → $\sqrt[x]{y}$

Affected by Flags: Numerical Results (−3)

Remarks: Note that while the *stack* syntax is y x XROOT (the root is the second argument), the *algebraic* syntax is XROOT(*x*, *y*) (the root is the first argument) for consistency with the EquationWriter application.

XROOT is equivalent to $y^{1/x}$, but with greater accuracy.

If $y < 0$, x must be an integer.

Related Commands: ^

XSEND

XModem Send Command: Sends a copy of the named object via XModem.

Level 1	→	Level 1
'*name*'	→	

Keyboard Access: ◀ I/O NXT XSEN

Affected by Flags: I/O Device (−33)

Remarks: A receiving HP 48 must execute XRECV to receive an object via XModem.

To start the transfer more quickly, start the receiving XModem *after* executing XSEND. Also, configuring the receiving modem *not* to do CRC checksums (if possible) will avoid a 30 to 60-second delay when starting the transfer.

If you are transferring data between two HP 48s, executing `« AAA BBB CCC » XSEND` sends *AAA*, *BBB*, and *CCC*. You also need to use a list on the receiving end (`« AAA BBB CCC » XRECV`, for example).

Related Commands: BAUD, RECN, RECV, SEND, XRECV

XVOL

X Volume Coordinates Command: Sets the width of the view volume in the reserved variable *VPAR*.

{ }

Level 2	Level 1	→	Level 1
x_{left}	x_{right}	→	

Keyboard Access: ↰ PLOT NXT 3D VPAR XVOL

Affected by Flags: None

Remarks: x_{left} and x_{right} set the x-coordinates for the view volume used in 3D plots. These values are stored in the reserved variable *VPAR*. See appendix D, "Reserved Variables," for more information about *VPAR*.

Related Commands: EYEPT, XXRNG, YVOL, YYRNG, ZVOL

XXRNG

X Range of an Input Plane (Domain) Command: Specifies the x range of an input plane (domain) for GRIDMAP and PARSURFACE plots.

{ }

Level 2	Level 1	→	Level 1
x_{min}	x_{max}	→	

Keyboard Access: ↰ PLOT NXT 3D VPAR XXRN

Affected by Flags: None

Remarks: x_{min} and x_{max} are real numbers that set the x-coordinates for the input plane. These values are stored in the reserved variable *VPAR*. See appendix D, "Reserved Variables," for more information about *VPAR*.

Related Commands: EYEPT, NUMX, NUMY, XVOL, YVOL, YYRNG, ZVOL

ΣX*Y

Sum of x Times y Command: Sums the products of each of the corresponding values in the independent- and dependent-variable columns of the current statistical matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{sum}

Keyboard Access: ↩ STAT SUMS Σx*y

Affected by Flags: None

Remarks: The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable ΣPAR. The default independent-variable column number is 1.

The dependent-variable column is specified by YCOL and is stored as the second parameter in reserved variable ΣPAR. The default dependent-variable column number is 2.

Related Commands: NΣ, ΣX, XCOL, ΣX^2, ΣY, ΣY^2

ΣY

Sum of y-Values Command: Sums the values in the dependent variable column of the current statistical matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{sum}

Keyboard Access: ↩ STAT SUMS Σy

Affected by Flags: None

Remarks: The dependent variable column is specified by YCOL, and is stored as the second parameter in the reserved variable ΣPAR. The default dependent variable column number is 2.

Related Commands: NΣ, ΣX, XCOL, ΣX*Y, ΣX^2, YCOL, ΣY^2

ΣY^2

Sum of Squares of y-Values Command: Sums the squares of the values in the dependent variable column of the current statistical matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{sum}

Keyboard Access: ↩ STAT SUMS ΣY^2

Affected by Flags: None

Remarks: The dependent variable column is specified by YCOL. The default dependent variable column number is 2.

Related Commands: NΣ, ΣX, XCOL, ΣX*Y, ΣX^2, YCOL, ΣY

YCOL

Dependent Column Command: Specifies the dependent variable column of the current statistics matrix (reserved variable ΣDAT).

{ }

Level 1	→	Level 1
n_{col}	→	

YCOL

Keyboard Access: ◀ STAT 3:PAR YCOL

Affected by Flags: None

Remarks: The dependent variable column number is stored as the second parameter in the reserved variable ΣPAR. The default dependent variable column number is 2.

YCOL will accept a noninteger real number and store it in ΣPAR, but subsequent commands that utilize the YCOL specification in ΣPAR will cause an error.

Related Commands: BARPLOT, BESTFIT, COLΣ, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, XCOL

YRNG

y-Axis Display Range Command: Specifies the y-axis display range.

{}

Level 2	Level 1	→	Level 1
y_{min}	y_{max}	→	

Keyboard Access: ◀ PLOT PPAR YRNG

Affected by Flags: None

Remarks: The y-axis display range is stored in the reserved variable $PPAR$ as y_{min} and y_{max} in the complex numbers (x_{min}, y_{min}) and (x_{max}, y_{max}). These complex numbers are the first two elements of $PPAR$ and specify the coordinates of the lower left and upper right corners of the display ranges.

The default values of y_{min} and y_{max} are -3.1 and 3.2, respectively.

Related Commands: AUTO, PDIM, PMAX, PMIN, XRNG

YSLICE

Y-Slice Plot Command: Sets the plot type to YSLICE.

Keyboard Access: ◀ PLOT NXT 3D PTYPE YSLIC

Affected by Flags: None

Remarks: When plot type is set YSLICE, the DRAW command plots a slicing view of a scalar function of two variables. YSLICE requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

{ x_{left} x_{right} y_{near} y_{far} z_{low} z_{high} x_{min} x_{max} y_{min} y_{max} x_{eye} y_{eye} z_{eye} x_{step} y_{step} }

For plot type YSLICE, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} determines the interval between plotted x-values within each "slice".
- y_{step} determines the number of slices to draw.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

For plot type YSLICE, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is not used.
- (x_{max}, y_{max}) is not used.

YSLICE

- *indep* is a name specifying the independent variable. The default value of *indep* is X.
- *res* is a real number specifying the interval, in user-unit coordinates, between plotted values of the independent variable; or a binary integer specifying the interval in pixels. The default value is #0, which specifies an interval of 1 pixel.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command YSLICE places YSLICE in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is Y.

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME

YVOL

Y Volume Coordinates Command: Sets the depth of the view volume in the reserved variable *VPAR*.

{ }

Level 2	Level 1	→	Level 1
y_{near}	y_{far}	→	

Keyboard Access: ◀ PLOT NXT 3D VPAR YVOL

Affected by Flags: None

Remarks: The variables y_{near} and y_{far} are real numbers that set the y-coordinates for the view volume used in 3D plots. y_{near} must be less than y_{far}. These values are stored in the reserved variable *VPAR*.

Related Commands: EYEPT, XVOL, XXRNG, YYRNG, ZVOL

YYRNG

Y Range of an Input Plane (Domain) Command: Specifies the y range of an input plane (domain) for GRIDMAP and PARSURFACE plots.

{ }

Level 2	Level 1	→	Level 1
y_{near}	y_{far}	→	

Keyboard Access: ◀ PLOT NXT 3D WFRM YYRN

Affected by Flags: None

Remarks: The variables $y_{y\ near}$ and $y_{y\ far}$ are real numbers that set the y-coordinates for the input plane. These values are stored in the reserved variable VPAR.

Related Commands: EYEPT, XVOL, XXRNG, YVOL, ZVOL

ZFACTOR

Gas Compressibility Z Factor Function: Calculates the gas compressibility correction factor for nonideal behavior of a hydrocarbon gas.

{ }

Level 2	Level 1	→	Level 1
x_{Tr}	y_{Pr}	→	$x_{Zfactor}$
x_{Tr}	'*symb*'	→	'ZFACTOR(x_{Tr},*symb*)'
'*symb*'	y_{Pr}	→	'ZFACTOR(*symb*,y_{Pr})'
'$symb_1$'	'$symb_2$'	→	'ZFACTOR($symb_1$,$symb_2$)'

Keyboard Access: ◀ EQ LIB UTILS ZFCT

ZFACTOR

Affected by Flags: Numerical Results (−3)

Remarks: x_{Tr} is the reduced temperature: the ratio of the actual temperature (T) to the pseudocritical temperature (T_c). (Calculate the ratio using absolute temperatures.) x_{Tr} must be between 1.05 and 3.0.

y_{Pr} is the reduced pressure: the ratio of the actual pressure (P) to the pseudocritical pressure (P_c). y_{Pr} must be between 0 and 30.

x_{Tr} and y_{Pr} must be real numbers or unit objects that reduce to dimensionless numbers.

ZVOL

Z Volume Coordinates Command: Sets the height of the view volume in the reserved variable $VPAR$.

{ }

Level 2	Level 1	→	Level 1
x_{low}	x_{high}	→	

Keyboard Access: ↩ PLOT NXT 3D VPAR ZVOL

Affected by Flags: None

Remarks: x_{low} and x_{high} are real numbers that set the z-coordinates for the view volume used in 3D plots. These values are stored in the reserved variable $VPAR$.

Related Commands: EYEPT, XVOL, XXRNG, YVOL, YYRNG

+

+

Add Analytic Function: Returns the sum of the arguments.

Level 2	Level 1	→	Level 1
z_1	z_2	→	z_1+z_2
[*array*]$_1$	[*array*]$_2$	→	[*array*]$_{1??+??2}$
z	'*symb*'	→	'*z*+(*symb*)'
'*symb*'	*z*	→	'*symb*+*z*'
'*symb*$_1$'	'*symb*$_2$'	→	'*symb*$_1$+*symb*$_2$'
{ *list*$_1$ }	{ *list*$_2$ }	→	{ *list*$_1$ *list*$_2$ }
obj$_A$	{ *obj*$_1$ ... *obj*$_n$ }	→	{ *obj*$_A$ *obj*$_1$ ... *obj*$_n$ }
{ *obj*$_1$ ... *obj*$_n$ }	*obj*$_A$	→	{ *obj*$_1$ ... *obj*$_n$ *obj*$_A$ }
"*string*$_1$"	"*string*$_2$"	→	"*string*$_1$ *string*$_2$"
obj	"*string*"	→	"*obj string*"
"*string*"	*obj*	→	"*string obj*"
#*n*$_1$	*n*$_2$	→	#*n*$_3$
n$_1$	#*n*$_2$	→	#*n*$_3$
#*n*$_1$	#*n*$_2$	→	#*n*$_3$
x$_1$_*unit*$_1$	*y*_*unit*$_2$	→	(*x*$_2$+*y*)_*unit*$_2$
'*symb*'	*x*_*unit*	→	'*symb*+*x*_*unit*'
*x*_*unit*	'*symb*'	→	'*x*_*unit*+*symb*'
grob$_1$	*grob*$_2$	→	*grob*$_3$

Keyboard Access: ⊞

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: The sum of a real number a and a complex number $(x,\ y)$ is the complex number $(x+a,\ y)$.

The sum of two complex numbers $(x_1,\ y_1)$ and $(x_2,\ y_2)$ is the complex number $(x_1+x_2,\ y_1+y_2)$.

+

The sum of a real array and a complex array is a complex array, where each element x of the real array is treated as a complex element $(x, 0)$. The arrays must have the same dimensions.

The sum of a binary integer and a real number is a binary integer that is the sum of the two arguments, truncated to the current wordsize. (The real number is converted to a binary integer before the addition.)

The sum of two binary integers is truncated to the current binary integer wordsize.

The sum of two unit objects is a unit object with the same dimensions as the level 1 argument. The units of the two arguments must be consistent.

The sum of two graphics objects is the same as the result of performing a logical OR, except that the two graphics objects *must* have the same dimensions.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The calculator assumes that the simple temperature units x_°C and x_°F represent thermometer temperatures when used as arguments to the functions <, >, ≤, ≥, ==, and ≠. This means that, in order to do the calculation, the calculator will first convert any Celsius temperature to kelvins and any Fahrenheit temperature to Rankines. (For other functions or *compound* temperature units, such as x_°C/min, the calculator assumes temperature units represent temperature differences, so there is no additive constant involved, and hence no conversion.)

The arithmetic operators +, −, %CH, and %T treat temperatures as differences, without any additive constant, but require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

Examples: { 1 2 3 } { A B C } + returns { 1 2 3 A B C }.

5_ft 9_in + returns 69_in.

[[0 1][1 3]] [[2 1][0 1]] + returns [[2 2][1 4]].

'FIRST' 'SECOND' + returns 'FIRST+SECOND'.

Related Commands: $-$, $*$, $/$, $=$

Subtract Analytic Function: Returns the difference of the arguments: the object in level 1 is subtracted from the object in level 2.

{ }

Level 2	Level 1	→	Level 1
z_1	z_2	→	$z_1 - z_2$
[*array*]$_1$	[*array*]$_2$	→	[*array*]$_{1-2}$
z	'*symb*'	→	'*z*−*symb*'
'*symb*'	*z*	→	'*symb*−*z*'
'*symb*$_1$'	'*symb*$_2$'	→	'*symb*$_1$ − *symb*$_2$'
#n_1	n_2	→	#n_3
n_1	#n_2	→	#n_3
#n_1	#n_2	→	#n_3
x_1 _ *unit*$_1$	*y*_*unit*$_2$	→	$(x_2 - y)$_*unit*$_2$
'*symb*'	*x*_*unit*	→	'*symb*−*x*_*unit*'
*x*_*unit*	'*symb*'	→	'*x*_*unit*−*symb*'

Keyboard Access: ⊟

Affected by Flags: Numerical Results (−3)

Remarks: The difference of a real number a and a complex number (x, y) is $(x-a, y)$ or $(a-x, -y)$. The difference of two complex numbers (x_1, y_1) and (x_2, y_2) is (x_1-x_2, y_1-y_2).

The difference of a real array and a complex array is a complex array, where each element x of the real array is treated as a complex element $(x, 0)$. The two array arguments must have the same dimensions.

The difference of a binary integer and a real number is a binary integer that is the sum of the level 2 number and the two's

-

complement of the level 1 number. (The real number is converted to a binary integer before the subtraction.)

The difference of two binary integers is a binary integer that is the sum of the level 2 number and the two's complement of the level 1 number.

The difference of two unit objects is a unit object with the same dimensions as the level 1 object. The units of the two arguments must be consistent.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The calculator assumes that the simple temperature units `x_°C` and `x_°F` represent thermometer temperatures when used as arguments to the functions <, >, ≤, ≥, ==, and ≠. This means that, in order to do the calculation, the calculator will first convert any Celsius temperature to kelvins and any Fahrenheit temperature to Rankines. (For other functions or *compound* temperature units, such as `x_°C/min`, the calculator assumes temperature units represent temperature differences, so there is no additive constant involved, and hence no conversion.)

The arithmetic operators +, −, %CH, and %T treat temperatures as differences, without any additive constant, but require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

Example: `25_ft 8_in` − returns `292_in`.

`[[ 5 1 ][ 3 3 ]] [[ 2 1 ][ 0 1 ]]` − returns `[[ 3 0 ][ 3 2 ]]`.

`'TOTAL' 'PART'` − returns `'TOTAL-PART'`.

Related Commands: +, *, /, =

---

*

Multiply Analytic Function: Returns the product of the arguments.

{}

Level 2	Level 1	→	Level 1
z_1	z_2	→	$z_1 z_2$
[[*matrix*]]	[*array*]	→	[[*matrix* × *array*]]
z	[*array*]	→	[*z* × *array*]
[*array*]	*z*	→	[*array* × *z*]
z	'*symb*'	→	'*z* ∗ *symb*'
'*symb*'	*z*	→	'*symb* ∗ *z*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ ∗ $symb_2$'
$\#n_1$	n_2	→	$\#n_3$
n_1	$\#n_2$	→	$\#n_3$
$\#n_1$	$\#n_2$	→	$\#n_3$
x_unit	*y_unit*	→	xy_unit_x × $unit_y$
x	*y_unit*	→	*xy_unit*
x_unit	*y*	→	*xy_unit*
'*symb*'	*x_unit*	→	'*symb* ∗ *x_unit*'
x_unit	'*symb*'	→	'*x_unit* ∗ *symb*'

Keyboard Access: ⊠

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: The product of a real number a and a complex number $(x,\ y)$ is the complex number $(xa,\ ya)$.

The product of two complex numbers $(x_1\,,\ y_1)$ and $(x_2\,,\ y_2)$ is the complex number $(x_1\,x_2\ -\ y_1\,y_2\,,\ x_1\,y_2\ +\ x_2\,y_1)$.

The product of a real array and a complex array or number is a complex array. Each element x of the real array is treated as a complex element $(x,\ 0)$.

*

Multiplying a matrix (level 2) by an array (level 1) returns a matrix product. The matrix must have the same number of columns as the array in level 1 has rows (or elements, if it is a vector).

Although a vector is entered and displayed as a *row* of numbers, the HP 48 treats a vector as an $n \times 1$ matrix when multiplying matrices or computing matrix norms.

Multiplying a binary integer by a real number returns a binary integer that is the product of the two arguments, truncated to the current wordsize. (The real number is converted to a binary integer before the multiplication.)

The product of two binary integers is truncated to the current binary integer wordsize.

When multiplying two unit objects, the scalar parts and the unit parts are multiplied separately.

Related Commands: $+$, $-$, $/$, $=$

/

/

Divide Analytic Function: Returns the quotient of the arguments: the level 2 object divided by the level 1 object.

{}

Level 2	Level 1	→	Level 1
z_1	z_2	→	$z_1 \ / \ z_2$
[*array*]	[[*matrix*]]	→	[[$matrix^{-1}$ x *array*]]
[*array*]	*z*	→	[*array/z*]
z	'*symb*'	→	'*z/symb*'
'*symb*'	*z*	→	'*symb/z*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ / $symb_2$'
$\#n_1$	n_2	→	$\#n_3$
n_1	$\#n_2$	→	$\#n_3$
$\#n_1$	$\#n_2$	→	$\#n_3$
x_unit_1	y_unit_2	→	$(x/y)_unit_1$ / $unit_2$
x	*y_unit*	→	*(x/y)_1/unit*
x_unit	*y*	→	*(x/y)_unit*
'*symb*'	*x_unit*	→	'*symb/x_unit*'
x_unit	'*symb*'	→	'*x_unit/symb*'

Keyboard Access:

SOLVE

÷

Affected by Flags: Numerical Results (−3)

Remarks: A real number a divided by a complex number $(x,\ y)$ returns $\left(\frac{ax}{x^2+y^2}, -\frac{ay}{x^2+y^2}\right)$. A complex number $(x,\ y)$ divided by a real number a returns the complex number $(x/a\ ,\ y/a)$.

A complex number $(x_1\ ,\ y_1\ )$ divided by another complex number $(x_2\ ,\ y_2\ )$ returns this complex quotient:

/

$$\left(\frac{x_1x_2+y_1y_2}{{x_2}^2+{y_2}^2},\frac{y_1x_2-x_1y_2}{{x_2}^2+{y_2}^2}\right)$$

An array **B** divided by a matrix **A** solves the system of equations **AX=B** for **X**; that is, $\mathbf{X}=\mathbf{A}^{-1}\mathbf{B}$. This operation uses 15-digit internal precision, providing a more precise result than the calculation INV(A)*B. The matrix must be square, and must have the same number of columns as the array has rows (or elements, if the array is a vector).

A binary integer divided by a real or binary number returns a binary integer that is the integral part of the quotient. (The real number is converted to a binary integer before the division.) A divisor of zero returns # 0.

When dividing two unit objects, the scalar parts and the unit parts are divided separately.

Related Commands: +, −, *, =

---

^

Power Analytic Function: Returns the value of the level 2 object raised to the power of the level 1 object.

{ }

Level 2	Level 1	→	Level 1
w	*z*	→	w^z
z	'*symb*'	→	'*z*^(*symb*)'
'*symb*'	*z*	→	'(*symb*)^*z*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ ^($symb_2$)'
x_unit	*y*	→	x^y _$unit^y$
x_unit	'*symb*'	→	'(*x_unit*)^(*symb*)'

Keyboard Access: y^x

Affected by Flags: Principal Solution (−1), Numerical Results (−3)

Remarks: If either argument is complex, the result is complex.

The branch cuts and inverse relations for w^z are determined by this relationship:

$$w^z = \exp(z(\ln\ w))$$

Related Commands: EXP, ISOL, LN, XROOT

<

Less Than Function: Tests whether one object is less than another object.

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	0/1
#n_1	#n_2	→	0/1
"$string_1$"	"$string_2$"	→	0/1
x	'*symb*'	→	'*x*<*symb*'
'*symb*'	*x*	→	'*symb*<*x*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$<$symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'*symb*'	→	'*x_unit*<*symb*'
'*symb*'	*x_unit*	→	'*symb*<*x_unit*'

Keyboard Access: PRG TEST <

Affected by Flags: Numerical Results (−3)

Remarks: The function < returns a true test result (1) if the level 2 argument is less than the level 1 argument, or a false test result (0) otherwise.

<

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, < returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "less than" means numerically smaller (1 is less than 2). For real numbers, "less than" also means more negative (−2 is less than −1).

For strings, "less than" means alphabetically previous ("ABC" is less than "DEF"; "AAA" is less than "AAB"; "A" is less than "AA"). In general, characters are ordered according to their character codes. This means, for example, that "B" is less than "a", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent, and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Related Commands: ≤, >, ≥, ==, ≠

<

Less Than or Equal Function: Tests whether one object is less than or equal to another object.

Level 2	Level 1	→	Level 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
$"string_1"$	$"string_2"$	→	0/1
x	$'symb'$	→	$'x \leq symb'$
$'symb'$	x	→	$'symb \leq x'$
$'symb_1'$	$'symb_2'$	→	$'symb_1 \leq symb_2'$
x_unit_1	y_unit_2	→	0/1
x_unit	$'symb'$	→	$'x_unit \leq symb'$
$'symb'$	x_unit	→	$'symb \leq x_unit'$

Keyboard Access: (PRG) TEST ≤

Affected by Flags: Numerical Results (−3)

Remarks: The function ≤ returns a true test result (1) if the level 2 argument is less than or equal to the level 1 argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, ≤ returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "less than or equal" means numerically equal or smaller (1 is less than 2). For real numbers, "less than or equal" also means equally or more negative (−2 is less than −1).

For strings, "less than or equal" means alphabetically equal or previous ("ABC" is less than or equal to "DEF"; "AAA" is less than or equal to "AAB"; "A" is less than or equal to "AA"). In general, characters are ordered according to their character codes. This means, for example, that "B" is less than "a", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperature and not differences in temperature. For compound temperature units, the calculator assumes temperature units represent

≤

temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Related Commands: <, >, ≥, ==, ≠

>

Greater Than Function: Tests whether one object is greater than another object.

{ }

Level 2	Level 1	→	Level 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
$"string_1"$	$"string_2"$	→	0/1
x	$'symb'$	→	$'x>symb'$
$'symb'$	x	→	$'symb>x'$
$'symb_1'$	$'symb_2'$	→	$'symb_1>symb_2'$
x_unit_1	y_unit_2	→	0/1
x_unit	$'symb'$	→	$'x_unit>symb'$
$'symb'$	x_unit	→	$'symb>x_unit'$

Keyboard Access: PRG TEST >

Affected by Flags: Numerical Results (−3)

Remarks: The function > returns a true test result (1) if the level 2 argument is greater than the level 1 argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, > returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "greater than" means numerically greater (2 is greater than 1). For real numbers, "greater than" also means less negative (−1 is greater than −2).

For strings, "greater than" means alphabetically subsequent ("DEF" is greater than "ABC"; "AAB" is greater than "AAA"; "AA" is greater than "A"). In general, characters are ordered according to their character codes. This means, for example, that "a" is greater than "B", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperature. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Related Commands: $<$, $\leq$, $\geq$, $==$, $\neq$

≥

Greater Than or Equal Function: Tests whether one object is greater than or equal to another object.

{ }

Level 2	Level 1	→	Level 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
"$string_1$"	"$string_2$"	→	0/1
x	'$symb$'	→	'$x \geq symb$'
'$symb$'	x	→	'$symb \geq x$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \geq symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'$symb$'	→	'$x_unit \geq symb$'
'$symb$'	x_unit	→	'$symb \geq x_unit$'

Keyboard Access: PRG TEST ≥

Affected by Flags: Numerical Results (−3)

≥

Remarks: The function ≥ returns a true test result (1) if the level 2 argument is greater than or equal to the level 1 argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, ≥ returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "greater than or equal to" means numerically equal or greater (2 is greater than or equal to 1). For real numbers, "greater than or equal to" also means equally or less negative (−1 is greater than or equal to −2).

For strings, "greater than or equal to" means alphabetically equal or subsequent ("DEF" is greater than or equal to "ABC"; "AAB" is greater than or equal to "AAA"; "AA" is greater than or equal to "A"). In general, characters are ordered according to their character codes. This means, for example, that "a" is greater than or equal to "B", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperature. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Related Commands: <, ≤, >, ==, ≠

=

Equals Analytic Function: Returns an equation formed from the two arguments.

Level 2	Level 1	→	Level 1
z_1	z_2	→	'$z_1 = z_2$'
z	'$symb$'	→	'z=$symb$'
'$symb$'	z	→	'$symb$=z'
'$symb_1$'	'$symb_2$'	→	'$symb_1 = symb_2$'
y	x_unit	→	'y=x_unit'
y_unit	x	→	'y_unit=x'
y_unit	x_unit	→	'y_unit=x_unit'
'$symb$'	x_unit	→	'$symb$=x_unit'
x_unit	'$symb$'	→	'x_unit=$symb$'

Keyboard Access: ↩ =

Affected by Flags: Numerical Results (−3)

Remarks: The equals sign equates two expressions such that the difference between the two expressions is zero.

In Symbolic Results mode, the result is an algebraic equation. In Numerical Results mode, the result is the difference of the two arguments because = acts equivalent to −. This allows expressions and equations to be used interchangeably as arguments for symbolic and numerical rootfinders.

The numerical evaluation of an equation using the HP Solve application implicitly involves the subtraction of terms. See the entry for "−" for information about the effects of subtraction.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The arithmetic operators +, −, %, %CH, and %T treat temperatures as differences, without any additive constant. However, +, −, %CH, and %T require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

=

Related Commands: DEFINE, EVAL, –

==

Logical Equality Function: Tests if two objects are equal.

{ }

Level 2	Level 1	→	Level 1
obj_1	obj_2	→	0/1
(x,0)	x	→	0/1
x	(x,0)	→	0/1
z	'symb'	→	'z==symb'
'symb'	z	→	'symb==z'
$'symb_1'$	$'symb_2'$	→	$'symb_1==symb_2'$

Keyboard Access: PRG TEST ==

Affected by Flags: Numerical Results (–3)

Remarks: The function == returns a true result (1) if the two objects are the same type and have the same value, or a false result (0) otherwise. Lists and programs are considered to have the same values if the objects they contain are identical.

If one object is algebraic (or a name), and the other is a number (real or complex) or an algebraic, == returns a symbolic comparison expression that can be evaluated to return a test result.

Note that == is used for comparisons, while = separates two sides of an equation.

If the imaginary part of a complex number is 0, it is ignored when the complex number is compared to a real number, so, for example, 6 (6,0) == returns 1.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent

temperatures and not differences in temperature. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Related Commands: SAME, TYPE, <, ≤, >, ≥, ≠

≠

Not Equal Function: Tests if two objects are not equal.

{ }

Level 2	Level 1	→	Level 1
obj_1	obj_2	→	0/1
(*x*,0)	*x*	→	0/1
x	(*x*,0)	→	0/1
z	'*symb*'	→	'*z* ≠ *symb*'
'*symb*'	*z*	→	'*symb* ≠ *z*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ ≠ $symb_2$'

Keyboard Access: PRG TEST ≠

Affected by Flags: Numerical Results (−3)

Remarks: The function ≠ returns a true result (1) if the two objects have different values, or a false result (0) otherwise. (Lists and programs are considered to have the same values if the objects they contain are identical.)

If one object is algebraic or a name, and the other is a number, a name, or algebraic, ≠ returns a symbolic comparison expression that can be evaluated to return a test result.

If the imaginary part of a complex number is 0, it is ignored when the complex number is compared to real number, so, for example, 5 (5,0) ≠ returns 0.

≠

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Related Commands: SAME, TYPE, <, ≤, >, ≥, ==

!

Factorial (Gamma) Function: Returns the factorial $n!$ of a positive integer argument n, or the gamma function $\Gamma(x+1)$ of a non-integer argument x.

{ }

Level 1	→	Level 1
n	→	*n*!
x	→	Γ(*x*+1)
'*symb*'	→	'(*symb*!)'

Keyboard Access: MTH NXT PROB !

Affected by Flags: Numerical Results (−3), Underflow Exception (−20), Overflow Exception (−21)

Remarks: For $x \geq 253.1190554375$ or $n < 0$, ! causes an Overflow exception (if flag −21 is set, the exception is treated as an error). For non-integer $x \leq -254.1082426465$, ! causes an Underflow exception (if flag −20 is set, the exception is treated as an error).

In algebraic syntax, ! follows its argument. Thus the algebraic syntax for the factorial of 7 is '7!'.

For non-integer arguments x, $x! = \Gamma(x+1)$, defined for $x > -1$ as this:

$$\Gamma(x+1) = \int_0^\infty e^{-t} t^x \, dt$$

and defined for other values of x by analytic continuation:

$$\Gamma(x+1) = n \cdot \Gamma(x)$$

Related Commands: COMB, PERM

∫

Integral Function: Integrates an *integrand* from *lower limit* to *upper limit* with respect to a specified variable of integration.

{ }

Level 4	Level 3	Level 2	Level 1	→	Level 1
lower limit	*upper limit*	*integrand*	'*name*'	→	'$symb_{integral}$'

Keyboard Access: → ∫

Affected by Flags: Numerical Results (−3), Number Format (−45 to −50)

Remarks: The algebraic syntax for ∫ parallels its stack syntax:

∫(*lower limit, upper limit, integrand, name*)

where *lower limit*, *upper limit*, and *integrand* can be real or complex numbers, unit objects, names, or algebraic expressions.

Evaluating ∫ in Symbolic Results mode (flag −3 *clear*) returns a symbolic result to level 1. The HP 48 does symbolic integration by *pattern matching*. The HP 48 can integrate the following:

- All built-in functions whose antiderivatives can be expressed in terms of other built-in functions—for example, SIN can be integrated since its antiderivative COS is a built-in function. The arguments for these functions must be linear.

- Sums, differences, and negations of built-in functions whose antiderivatives can be expressed in terms of other built-in functions—for example, 'SIN(X)-COS(X)'.
- Derivatives of all built-in functions—for example, 'INV(1+X^2)' can be integrated because it is the derivative of the built-in function ATAN.
- Polynomials whose base term is linear—for example, 'X^3+X^2-2*X+6' can be integrated since X is a linear term. '(X^2-6)^3+(X^2-6)^2' cannot be integrated since X^2-6 is not linear.
- Selected patterns composed of functions whose antiderivatives can be expressed in terms of other built-in functions—for example, '1/(COS(X)*SIN(X))' returns 'LN(TAN(X))'.

If the result of the integration is an expression with no integral sign in the result, the symbolic integration was successful. If, however, the result still contains an integral sign, try rearranging the expression and evaluating again, or estimate the answer using numerical integration.

A successful result of symbolic integration has this form:

'*result*|(*name*=*upper limit*)−(*result*|(*name*=*lower limit*))'

See the | (where) entry for more information about its functionality. A second evaluation substitutes the limits of integration into the variable of integration, completing the procedure.

Evaluating ∫ in Numerical Results mode (flag −3 *set*) returns a numerical approximation, and stores the error of integration in variable *IERR*. ∫ consults the number format setting to determine how accurately to compute the result.

Examples: In Symbolic Results mode (flag −3 clear) this command sequence:

```
1 2 '10*X' 'X' ∫
```

returns

```
'10*(X^2/2)|(X=2)−(10*(X^2/2)|(X=1))'
```

Subsequent evaluation substitutes the limits of integration, and returns 15.

In Numeric Results mode (flag −3 set) the above command sequence returns the numerical approximation 15. In addition, the variable *IERR* is created, and contains the error of integration .000000000015.

Related Commands: TAYLR, ∂, Σ

∂

Derivative Function: Takes the derivative of an expression, number, or unit object with respect to a specified variable of differentiation.

{ }

Level 2	Level 1	→	Level 1
'$symb_1$'	'*name*'	→	'$symb_2$'
z	'*name*'	→	0
x_unit	'*name*'	→	0

Keyboard Access: → ∂

Affected by Flags: Numerical Results (−3)

Remarks: When executed in stack syntax, ∂ executes a *complete* differentiation: the expression '$symb_1$' is evaluated repeatedly until it contains no derivatives. As part of this process, if the variable of differentiation *name* has a value, the final form of the expression substitutes that value substituted for all occurrences of the variable.

The algebraic syntax for ∂ is '∂*name*(*symb*$_1$)'. When executed in algebraic syntax, ∂ executes a *stepwise* differentiation of $symb_1$, invoking the chain rule of differentiation—the result of one evaluation of the expression is the derivative of the argument expression $symb_1$, multiplied by a new subexpression representing the derivative of $symb_1$'s argument.

If ∂ is applied to a function for which the HP 48 does not provide a derivative, ∂ returns a new function whose name is *der* followed by the original function name.

∂

Example: In Radians mode, the command sequence `'∂X(SIN(Y))'` EVAL returns `'COS(Y)*∂X(Y)'`.

When Y has the value `X^2`, the command sequence `'SIN(Y)' 'X' ∂` returns `'COS(X^2)*(2*X)'`. The differentiation has been executed in stack syntax, so that all of the steps of differentiation have been carried out in a single operation.

Related Commands: TAYLR, $\int$, Σ

%

Percent Function: Returns x (level 2) percent of y (level 1).

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	*xy/100*
x	'*symb*'	→	'%(*x*,*symb*)'
'*symb*'	*x*	→	'%(*symb*,*x*)'
'$symb_1$'	'$symb_2$'	→	'%($symb_1$, $symb_2$)'
x	*y_unit*	→	(*xy/100*)_*unit*
x_unit	*y*	→	(*xy/100*)_*unit*
'*symb*'	*x_unit*	→	'%(*symb*,*x_unit*)'
x_unit	'*symb*'	→	'%(*x_unit*,*symb*)'

Keyboard Access: MTH REAL %

Affected by Flags: Numerical Results (−3)

Remarks: Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The arithmetic operators +, −, %, %CH, and %T treat temperatures as differences, without any additive constant. However, +, −, %CH, and %T require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

For more information on using temperature units with arithmetic functions, see the entry for +.

Example: 23.7 995 % returns 235.815.

15 176_kg % returns 26.4_kg.

100_°C 50 % returns 50_°C.

Related Commands: %CH, %T

π

π Function: Returns the symbolic constant 'π' or its numerical representation, 3.14159265359.

Level 1	→	Level 1
	→	'π'
	→	3.14159265359

Keyboard Access: ◀ π

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

Evaluating π returns its numerical representation if flag −2 or −3 is set; otherwise, returns its symbolic representation.

Remarks: The number returned for π is the closest approximation of the constant π to 12-digit accuracy.

In Radians mode with flags −2 and −3 clear (to return symbolic results), trigonometric functions of π and π/2 are automatically simplified. For example, evaluating 'SIN(π)' returns zero. However, if flag −2 or flag −3 is set (to return numerical results),

π

then evaluating '`SIN(π)`' returns the numerical approximation `-2.06761537357E-13`.

Related Commands: e, i, MAXR, MINR, $\rightarrow Q\pi$

Σ

Summation Function: Calculates the value of a finite series.

{ }

Level 4	Level 3	Level 2	Level 1	→	Level 1
'*indx*'	x_{init}	x_{final}	*smnd*	→	x_{sum}
'*indx*'	'*init*'	x_{final}	*smnd*	→	'$\Sigma(indx{=}init,x_{final},smnd)$'
'*indx*'	x_{init}	'*final*'	*smnd*	→	'$\Sigma(indx{=}x_{init},final,smnd)$'
'*indx*'	'*init*'	'*final*'	*smnd*	→	'$\Sigma(indx{=}init,final,smnd)$'

Keyboard Access: → Σ

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

Remarks: The summand argument *smnd* can be a real number, a complex number, a unit object, a local or global name, or an algebraic object.

The algebraic syntax for Σ differs from the stack syntax. The algebraic syntax is this:

'Σ(*index=initial,final,summand*)'

Examples: In Symbolic Results mode (flags −2 and −3 clear), the command sequence '`N`' `1` `5` '`A^N`' `Σ` returns '`A+A^2+A^3+A^4+A^5`'.

The command sequence '`N`' `1` '`M`' '`A^N`' `Σ` returns '`Σ(N=1,M,A^N)`'.

Related Commands: TAYLR, $\int$, ∂

Σ+

Sigma Plus Command: Adds one or more data points to the current statistics matrix (reserved variable ΣDAT).

Level m ... Level 2	Level 1	→ Level 1
	x	→
	$[\ x_1\ \ x_2 \ldots\ x_m\ ]$	→
	$[[\ x_{1\,1} \ldots\ x_{1\,m}\ ]\ [\ x_{n\,1} \ldots\ x_{n\,m}\ ]]$	→
$x_1 \ldots\ x_{m-1}$	x_m	→

Keyboard Access: ↰ STAT MATH Σ+

Affected by Flags: None

Remarks: For a statistics matrix with m columns, arguments for Σ+ can be entered several ways:

- To enter one data point with a single coordinate value, the argument for Σ+ must be a real number.
- To enter one data point with multiple coordinate values, the argument for Σ+ must be a vector of m real coordinate values.
- To enter several data points, the argument for Σ+ must be a matrix of n rows of m real coordinate values.

In each case, the coordinate values of the data point(s) are added as new rows to the current statistics matrix (reserved variable ΣDAT). If ΣDAT does not exist, Σ+ creates an $n \times m$ matrix and stores the matrix in ΣDAT. If ΣDAT does exist, an error occurs if it does not contain a real matrix, or if the number of coordinate values in each data point entered with Σ+ does not match the number of columns in the current statistics matrix.

Once ΣDAT exists, individual data points of m coordinates can be entered as m separate real numbers or an m-element vector.

LASTARG returns the m-element vector in either case.

Example: The sequence CLΣ [2 3 4] Σ+ 3 1 7 Σ+ creates the matrix [[2 3 4][3 1 7]] in ΣDAT.

Σ+

Related Commands: CLΣ, RCLΣ, STOΣ, Σ−

Σ−

Sigma Minus Command: Returns a vector of m real numbers (or one number x if $m = 1$) corresponding to the coordinate values of the last data point entered by Σ+ into the current statistics matrix (reserved variable ΣDAT).

	→	**Level 1**
	→	x
	→	$[\ x_1\ \ x_2\ \ \ldots\ \ x_m\ ]$

Keyboard Access: ↩ STAT DATA Σ−

Affected by Flags: None

Remarks: The last row of the statistics matrix is deleted.

The vector returned by Σ− can be edited or replaced, then restored to the statistics matrix by Σ+.

Related Commands: CLΣ, RCLΣ, STOΣ, Σ+

√

Square Root Analytic Function: Returns the (positive) square root of the argument.

√

{ }

Level 1	→	Level 1
z	→	$\sqrt{z}$
x_unit	→	$\sqrt{x}_u nit^1/2$
'*symb*'	→	$'\sqrt{(symb)}'$

Keyboard Access: [√x]

Affected by Flags: Principal Solution (−1), Numerical Results (−3)

Remarks: For a complex number (x_1, y_1), the square root is this complex number:

$$(x_2, y_2) = (\sqrt{r}\, cos\frac{\theta}{2}, \sqrt{r}\, sin\frac{\theta}{2})$$

where r = ABS (x_1, y_1), and θ = ARG (x_1, y_1).

If $(x_1, y_1) = (0, 0)$, then the square root is (0, 0).

The inverse of SQ is a *relation*, not a function, since SQ sends more than one argument to the same result. The inverse relation for SQ is expressed by ISOL as this *general solution*:

'≡1*√Z'

The function √ is the inverse of a *part* of SQ, a part defined by restricting the domain of SQ such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of SQ are called the *principal values* of the inverse relation. The √ function in its entirety is called the *principal branch* of the inverse relation, and the points sent by √ to the boundary of the restricted domain of SQ form the *branch cuts* of √.

The principal branch used by the HP 48 for √ was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued square root function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of √. The graph of the domain shows where the branch cut occurs: the heavy solid line marks one side of the cut, while the feathered lines mark the other side of

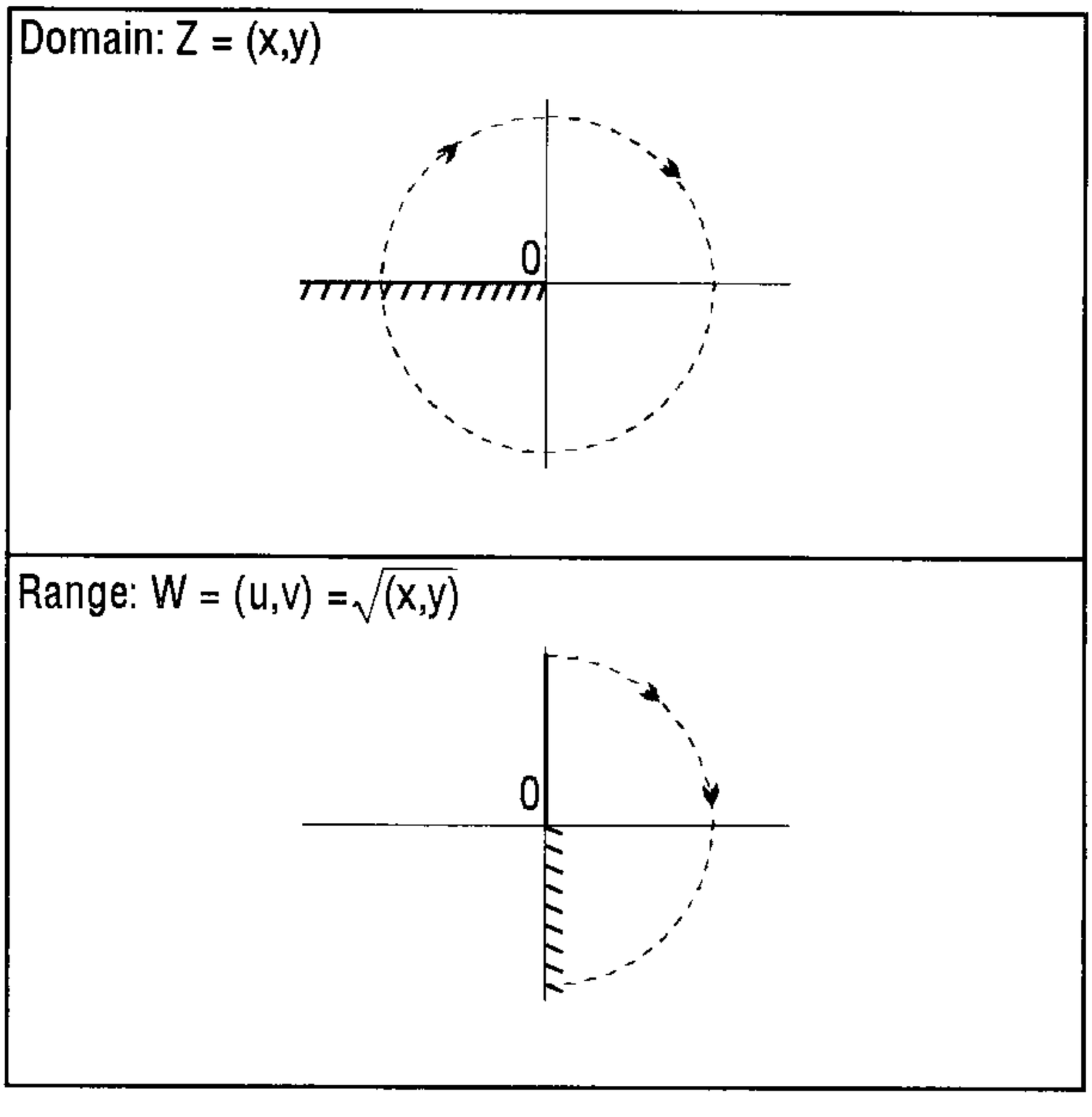
√

the cut. The graph of the range shows where each side of the cut is mapped under the function.

These graphs show the inverse relation '≡1*⌜Z' for the case *s1*=1. For the other value of *s1*, the half-plane in the lower graph is rotated. Taken together, the half-planes cover the whole complex plane, which is the domain of SQ.

View these graphs with domain and range reversed to see how the domain of SQ is restricted to make an inverse *function* possible. Consider the half-plane in the lower graph as the restricted domain Z = (*x*, *y*). SQ sends this domain onto the whole complex plane in the range W = (*u*, *v*) = SQ(*x*, *y*) in the upper graph.

Related Commands: SQ, ^, ISOL



_

Attach Unit Function: Attaches a unit expression to a real number. Performed automatically in the Unit Catalog (→ UNITS).

Level 2	Level 1	→	Level 1
x	*'unit expression'*	→	*x_unit*

Keyboard Access: → _

Affected by Flags: None

Related Commands: →UNIT

| (Where)

Where Function: Substitutes values for names in an expression.

Level 2	Level 1	→	Level 1
$'symb_{old}'$	{ $name_1$ $'symb_1'$ $name_2$ $'symb_2'$ ... }	→	$'symb_{new}'$
x	{ $name_1$ $'symb_1'$ $name_2$ $'symb_2'$ ... }	→	*x*
(*x*, *y*)	{ $name_1$ $'symb_1'$ $name_2$ $'symb_2'$ ... }	→	(*x*, *y*)

Keyboard Access: ← SYMBOLIC NXT |

Affected by Flags: Numerical Results (−3)

Remarks: | is used primarily in algebraic objects, where its syntax is:

$$'symb_{old} \mid \{ name_1 = symb_1 , \quad name_2 = symb_2 \quad \ldots \quad \}'$$

It enables algebraics to include variable-like substitution information about names. Symbolic functions that delay name evaluation (such as $\int$ and ∂) can then extract substitution information from local

variables and include that information in the expression, avoiding the problem that would occur if the local variables no longer existed when the local names were finally evaluated.

Related Commands: APPLY, QUOTE

→

Create Local Variables Command: Creates local variables.

Level n ... Level 1	→	Level 1
obj_1 ... obj_n	→	

Keyboard Access: 

Affected by Flags: None

Remarks: *Local variable structures* specify one or more local variables and a defining procedure.

A local variable structure consists of the → command, followed by one or more names, followed by a defining procedure—either a program or an algebraic. The → command stores objects from the stack into local variables with the specified names. The resultant *local variables* exist only while the defining procedure is being executed. The syntax of a local variable structure is one of the following:

- → $name_1$ $name_2$ ... $name_n$ « *program* »
- → $name_1$ $name_2$ ... $name_n$ ' *algebraic expression* '

Example: This program:

```
« → x y « x y * x y - + » »
```

takes an object from level 2 and stores it in local variable x, takes an object from level 1 and stores it in local variable y, and executes calculations with x and y in the defining procedure (in this case a program). When the defining procedure ends, local variables x and y disappear.

User-Defined Functions. A user-defined function is a variable containing a program that consists solely of a local variable structure.

For example, the variable *A*, containing this program:

```
« → x y z 'x*y^2+z' »
```

is a user-defined function. Like a built-in function, a user-defined function can take its arguments in stack syntax or algebraic syntax, and can take symbolic arguments. In addition, a user-defined function is differentiable if its defining procedure is an algebraic expression that contains only differentiable functions.

Related Commands: DEFINE, STO

4

Equation Reference

The Equation Library consists of 15 subjects and more than 100 titles. Each subject and title has a number that you can use with SOLVEQN to specify the set of equations. These numbers are shown in parentheses after the headings.

See the end of this section for references given in each subject (Reference: x). Remember that some equations are estimates and assume certain conditions. See the references or other standard texts for assumptions and limitations of the equations.

Solutions in the examples have been rounded to four decimal places.

Columns and Beams (1)

Variable Names and Descriptions

ϵ	Eccentricity (offset) of load
σcr	Critical stress
σmax	Maximum stress
θ	Slope at x
A	Cross-sectional area
a	Distance to point load
c	Distance to edge fiber (Eccentric Columns), or Distance to applied moment (beams)
E	Modulus of elasticity
I	Moment of inertia

Variable Names and Descriptions (continued)

K	Effective length factor of column
L	Length of column or beam
M	Applied moment
Mx	Internal bending moment at x
P	Load (Eccentric Columns), or Point load (beams)
Pcr	Critical load
r	Radius of gyration
V	Shear force at x
w	Distributed load
x	Distance along beam
y	Deflection at x

For simply supported beams and cantilever beams ("Simple Deflection" through "Cantilever Shear"), the calculations differ depending upon the location of x relative to the loads.

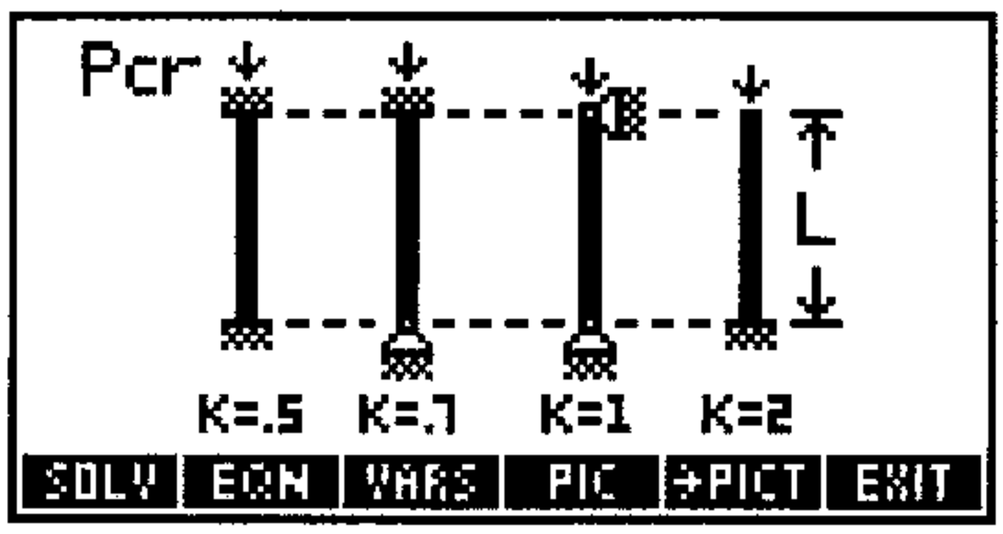
- Applied loads are positive downward.
- The applied moment is positive counterclockwise.
- Deflection is positive upward.
- Slope is positive counterclockwise.
- Internal bending moment is positive counterclockwise on the left-hand part.
- Shear force is positive downward on the left-hand part.

Reference: 2.

4-2 Equation Reference

Elastic Buckling (1, 1)

These equations apply to a slender column ($K \cdot L/r > 100$) with length factor K.



Equations:

$$\mathrm{Pcr} = \frac{\pi^2 \cdot \mathrm{E} \cdot \mathrm{A}}{\left(\frac{\mathrm{K} \cdot \mathrm{L}}{\mathrm{r}}\right)^2} \qquad \mathrm{Pcr} = \frac{\pi^2 \cdot \mathrm{E} \cdot \mathrm{I}}{\left(\mathrm{K} \cdot \mathrm{L}\right)^2} \qquad \sigma\mathrm{cr} = \frac{\mathrm{Pcr}}{\mathrm{A}} \qquad \mathrm{r} = \sqrt{\frac{\mathrm{I}}{\mathrm{A}}}$$

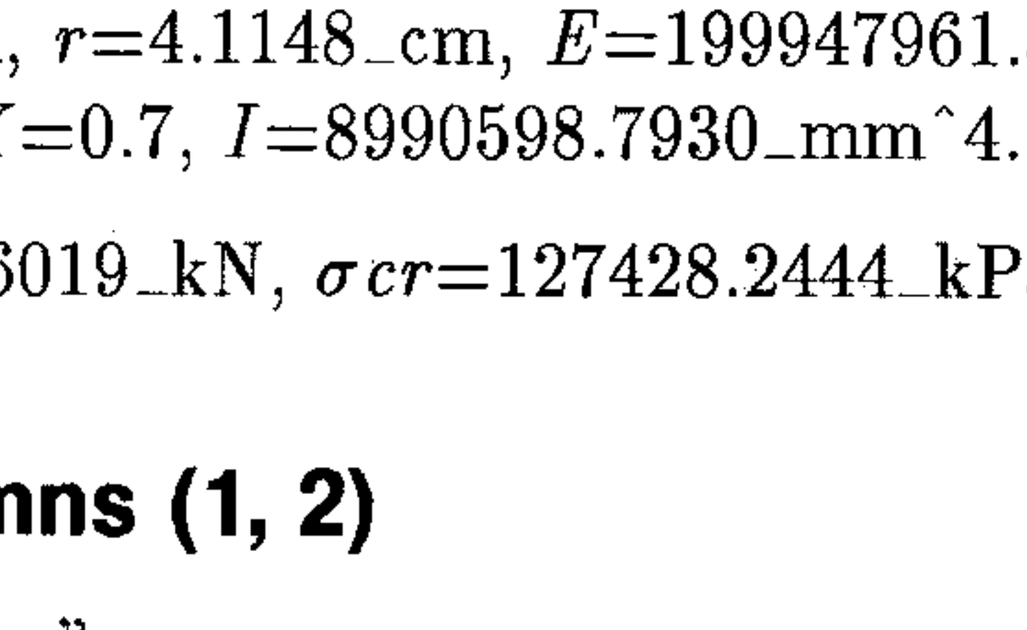
Example:

Given: L=7.3152_m, r=4.1148_cm, E=199947961.502_kPa, A=53.0967_cm^2, K=0.7, I=8990598.7930_mm^4.

Solution: Pcr=676.6019_kN, σcr=127428.2444_kPa.

Eccentric Columns (1, 2)

See "Elastic Buckling."



Equations:

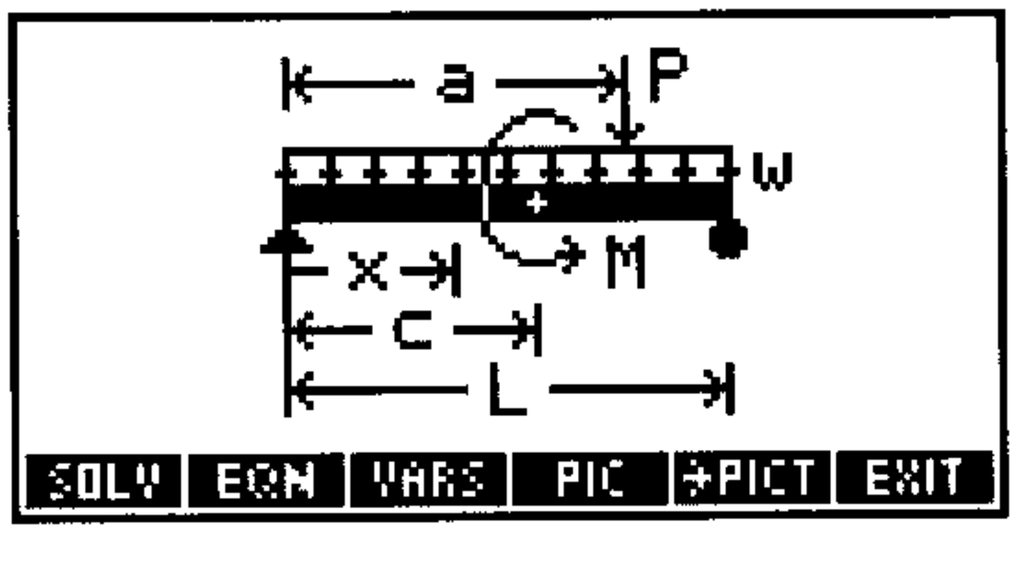
$$\sigma\max = \frac{P}{A} \cdot \left(1 + \frac{\epsilon \cdot c}{r^2} \cdot \left(\frac{1}{\cos\left(\frac{K \cdot L}{2 \cdot r} \cdot \sqrt{\frac{P}{E \cdot A}} \right)} \right) \right) \qquad r = \sqrt{\frac{I}{A}}$$

Example:

Given: L=6.6542_m, A=187.9351_cm^2, r=8.4836_cm, E=206842718.795_kPa, I=135259652.16_mm^4, K=1, P=1908.2571_kN, c=15.24_cm, ϵ=1.1806_cm.

Solution: σmax=140853.0970_kPa.

Simple Deflection (1, 3)



Equation:

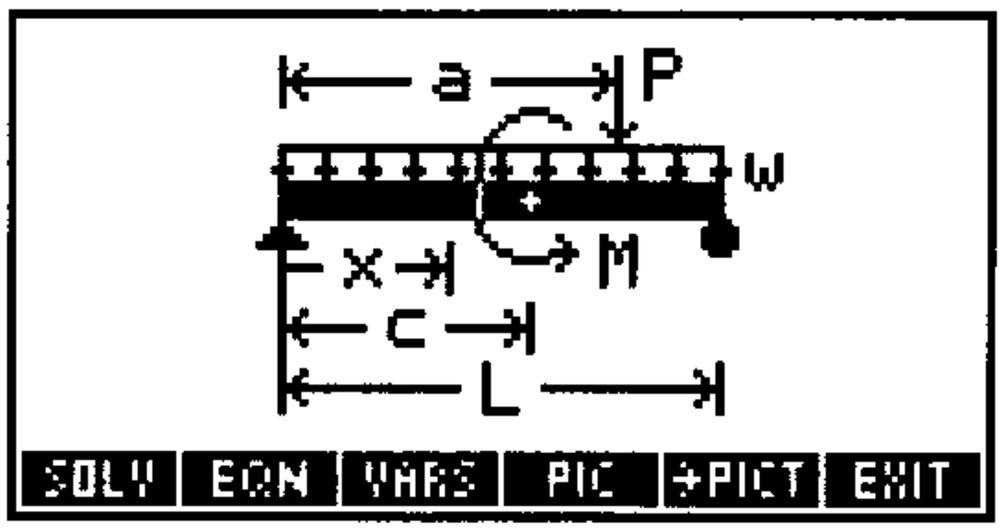
$$y = \frac{P \cdot \left(L - a \right) \cdot x}{6 \cdot L \cdot E \cdot I} \cdot \left(x^2 + \left(L - a \right)^2 - L^2 \right) - \frac{M \cdot x}{E \cdot I} \cdot \left(c - \frac{x^2}{6 \cdot L} - \frac{L}{3} - \frac{c^2}{2 \cdot L} \right) - \frac{w \cdot x}{24 \cdot E \cdot I} \cdot \left(L^3 + x^2 \cdot \left(x - 2 \cdot L \right) \right)$$

Example:

Given: L=20_ft, E=29000000_psi, I=40_in^4, a=10_ft, P=674.427_lbf, c=17_ft, M=3687.81_ft*lbf, w=102.783_lbf/ft, x=9_ft.

Solution: y=−.6005_in.

Simple Slope (1, 4)



Equation:

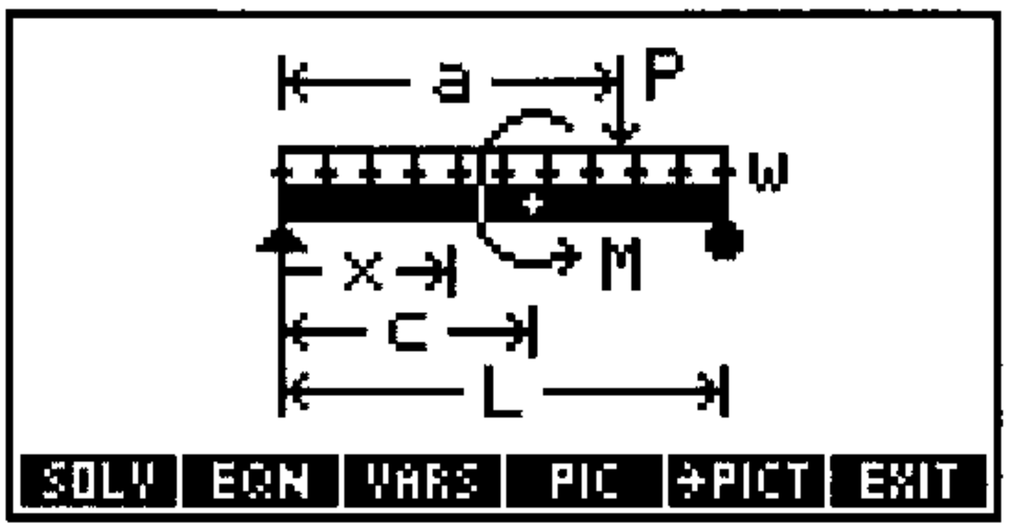
$$\Theta = \frac{P \cdot \left(L - a\right)}{6 \cdot L \cdot E \cdot I} \cdot \left(3 \cdot x^2 + \left(L - a\right)^2 - L^2\right) - \frac{M}{E \cdot I} \cdot \left(c - \frac{x^2}{2 \cdot L} - \frac{L}{3} - \frac{c^2}{2 \cdot L}\right) - \frac{w}{24 \cdot E \cdot I} \cdot \left(L^3 + x^2 \cdot \left(4 \cdot x - 6 \cdot L\right)\right)$$

Example:

Given: L=20_ft, E=29000000_psi, I=40_in^4, a=10_ft, P=674.427_lbf, c=17_ft, M=3687.81_ft*lbf, w=102.783_lbf/ft, x=9_ft.

Solution: Θ=−.0876_°.

Simple Moment (1, 5)



Equation:

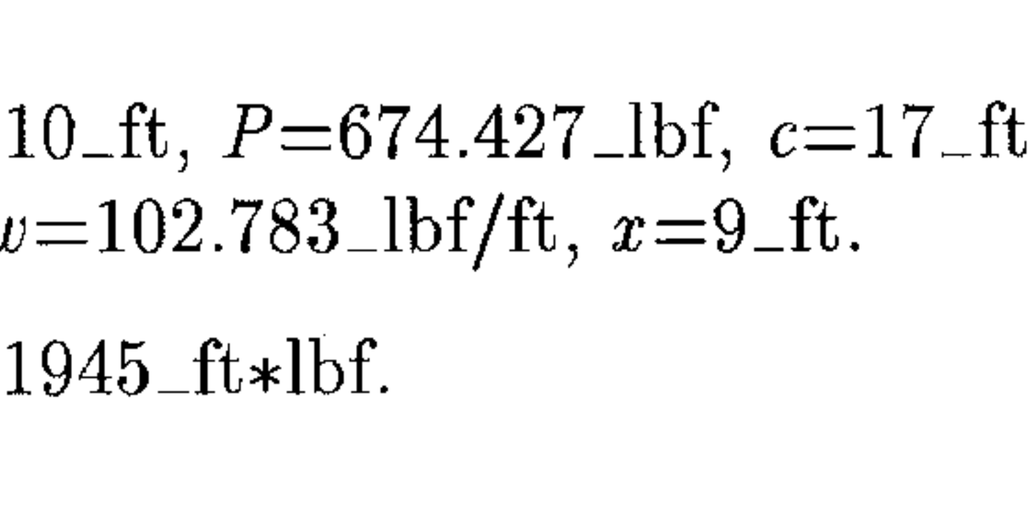
$$Mx = \frac{P \cdot \left(L - a\right) \cdot x}{L} + \frac{M \cdot x}{L} + \frac{w \cdot x}{2} \cdot \left(L - x\right)$$

Example:

Given: L=20_ft, a=10_ft, P=674.427_lbf, c=17_ft, M=3687.81_ft∗lbf, w=102.783_lbf/ft, x=9_ft.

Solution: Mx=9782.1945_ft∗lbf.

Simple Shear (1, 6)



Equation:

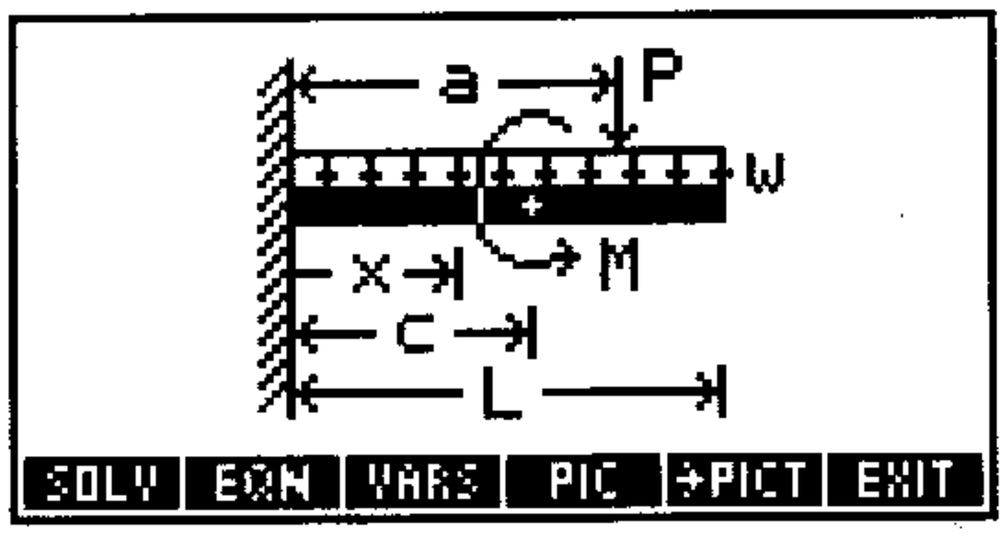
$$V = \frac{P \cdot \left(L - a\right)}{L} + \frac{M}{L} + \frac{w}{2} \cdot \left(L - 2 \cdot x\right)$$

Example:

Given: L=20_ft, a=10_ft, P=674.427_lbf, M=3687.81_ft*lbf, w=102.783_lbf/ft, x=9_ft.

Solution: V=624.387_lbf.

Cantilever Deflection (1, 7)



Equation:

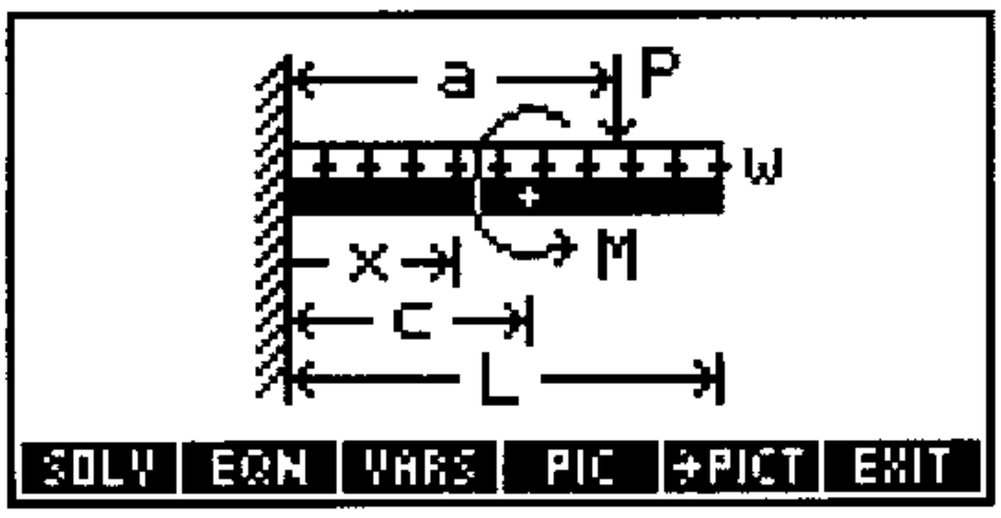
$$y = \frac{P \cdot x^2}{6 \cdot E \cdot I} \cdot \left(x - 3 \cdot a\right) + \frac{M \cdot x^2}{2 \cdot E \cdot I} - \frac{w \cdot x^2}{24 \cdot E \cdot I} \cdot \left(6 \cdot L^2 - 4 \cdot L \cdot x + x^2\right)$$

Example:

Given: L=10_ft, E=29000000_psi, I=15_in^4, P=500_lbf, M=800_ft*lbf, a=3_ft, c=6_ft, w=100_lbf/ft, x=8_ft.

Solution: y=−.3316_in.

Cantilever Slope (1, 8)



Equation:

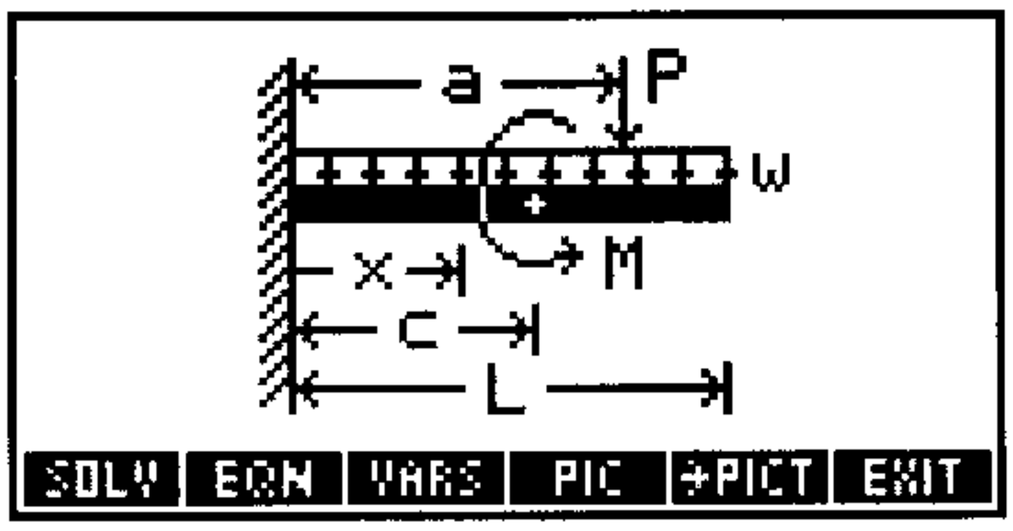
$$\Theta = \frac{P \cdot x}{2 \cdot E \cdot I} \cdot \left(x - 2 \cdot a \right) + \frac{M \cdot x}{E \cdot I} - \frac{w \cdot x}{6 \cdot E \cdot I} \cdot \left(3 \cdot L^2 - 3 \cdot L \cdot x + x^2 \right)$$

Example:

Given: L=10_ft, E=29000000_psi, I=15_in^4, P=500_lbf, M=800_ft*lbf, a=3_ft, c=6_ft, w=100_lbf/ft, x=8_ft.

Solution: Θ=−.2652_°.

Cantilever Moment (1, 9)



Equation:

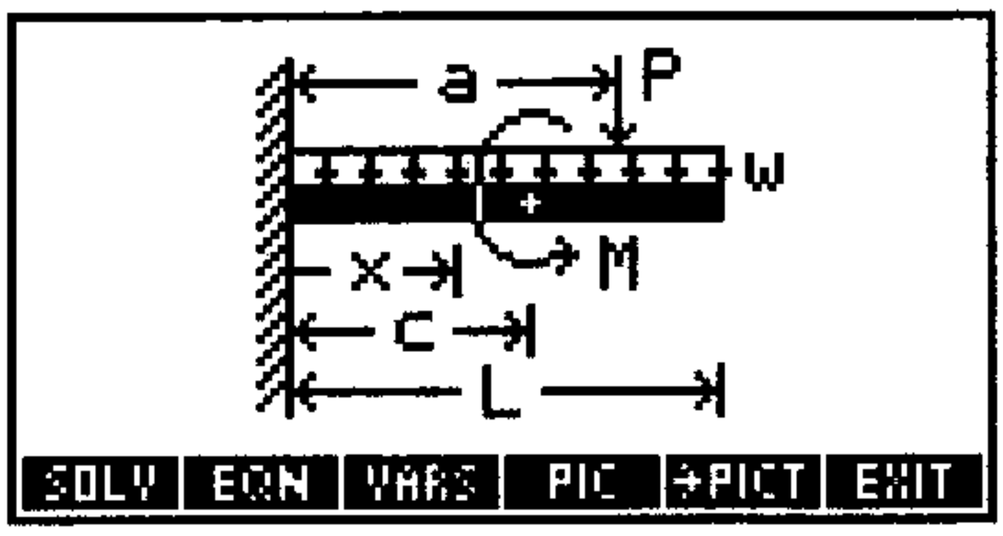
$$Mx = P \cdot \left(x - a \right) + M - \frac{W}{2} \cdot \left(L^2 - 2 \cdot L \cdot x + x^2 \right)$$

Example:

Given: L=10_ft, P=500_lbf, M=800_ft*lbf, a=3_ft, c=6_ft, w=100_lbf/ft, x=8_ft.

Solution: Mx=−200_ft*lbf.

Cantilever Shear (1, 10)



Equation:

$$V = P + w \cdot \left(L - x \right)$$

Example:

Given: L=10_ft, P=500_lbf, a=3_ft, x=8_ft, w=100_lbf/ft.

Solution: V=200_lbf.

Electricity (2)

Variable Names and Descriptions

ϵr	Relative permittivity
μr	Relative permeability
ω	Angular frequency
$\omega 0$	Resonant angular frequency
ϕ	Phase angle
$\phi p, \phi s$	Parallel and series phase angles
ρ	Resistivity

Variable Names and Descriptions (continued)

ΔI	Current change
Δt	Time change
ΔV	Voltage change
A	Wire cross-section area (Wire Resistance), or Solenoid cross-section area (Solenoid Inductance), or Plate area (Plate Capacitor)
$C, C1, C2$	Capacitance
Cp, Cs	Parallel and series capacitances
d	Plate separation
E	Energy
F	Force between charges
f	Frequency
$f0$	Resonant frequency
I	Current, or Total current (Current Divider)
$I1$	Current in R1
$Imax$	Maximum current
L	Inductance, or Length (Wire Resistance, Cylindrical Capacitor)
$L1, L2$	Inductance
Lp, Ls	Parallel and series inductances
N	Number of turns
n	Number of turns per unit length
P	Power
q	Charge
$q1, q2$	Point charge
Qp, Qs	Parallel and series quality factors
r	Charge distance
$R, R1, R2$	Resistance
ri, ro	Inside and outside radii
Rp, Rs	Parallel and series resistances

4-10 Equation Reference

Variable Names and Descriptions (continued)

t	Time
ti, tf	Initial and final times
V	Voltage, or Total voltage (Voltage Divider)
$V1$	Voltage across R1
Vi, Vf	Initial and final voltages
$Vmax$	Maximum voltage
XC	Reactance of capacitor
XL	Reactance of inductor

Reference: 3.

Coulomb's Law (2, 1)

This equation describes the electrostatic force between two charged particles.

Equation:

$$\mathrm{F} = \frac{1}{4 \cdot \pi \cdot \epsilon 0 \cdot \epsilon \mathrm{r}} \cdot \left(\frac{\mathrm{q1} \cdot \mathrm{q2}}{\mathrm{r}^2} \right)$$

Example:

Given: $q1$=1.6E−19_C, $q2$=1.6E−19_C, r=4.00E−13_cm, ϵr=1.00.

Solution: F=14.3801_N.

Ohm's Law and Power (2, 2)

Equations:

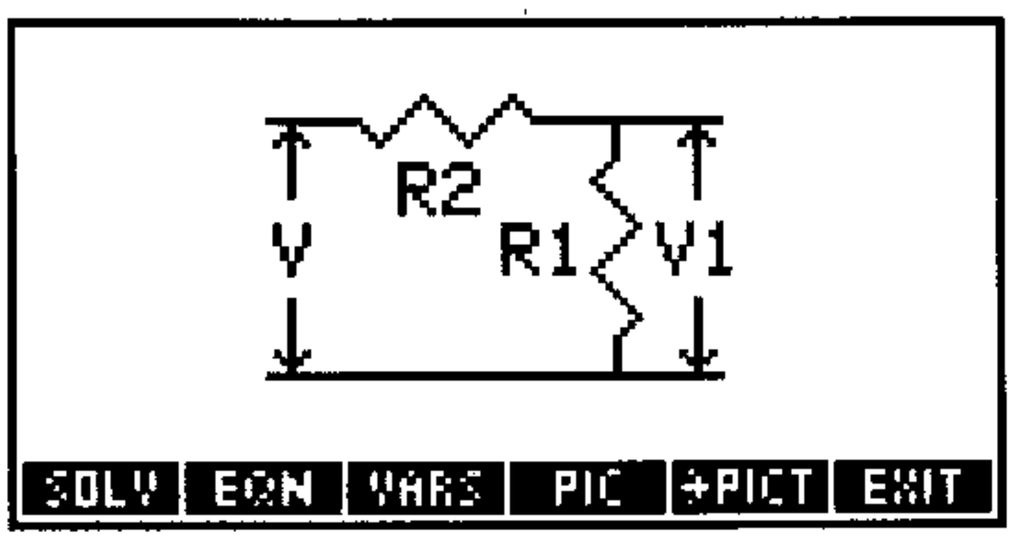
$$\mathrm{V} = \mathrm{I} \cdot \mathrm{R} \qquad \mathrm{P} = \mathrm{V} \cdot \mathrm{I} \qquad \mathrm{P} = \mathrm{I}^2 \cdot \mathrm{R} \qquad \mathrm{P} = \frac{\mathrm{V}^2}{\mathrm{R}}$$

Example:

Given: V=24_V, I=16_A.

Solution: R=1.5_Ω, P=384_W.

Voltage Divider (2, 3)



Equation:

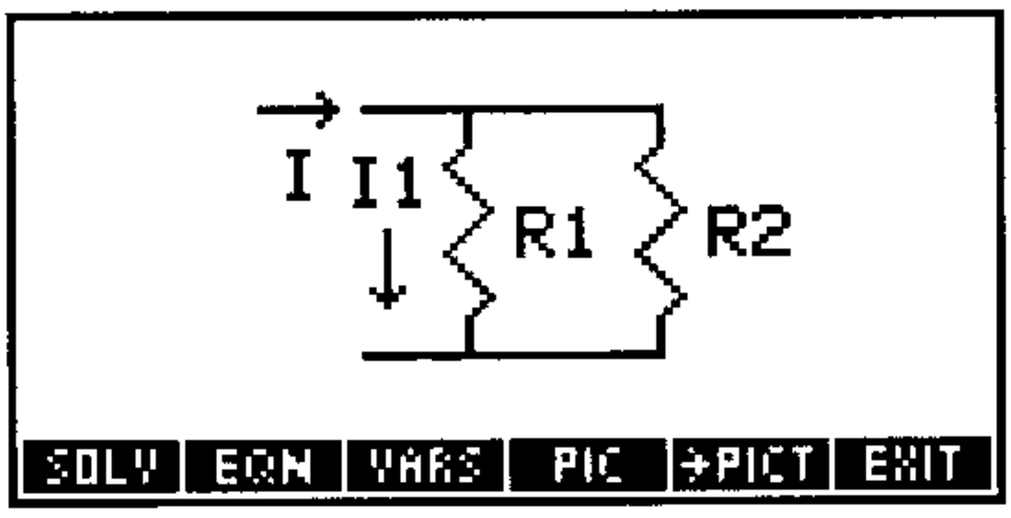
$$\mathrm{V1} = \mathrm{V} \cdot \left(\frac{\mathrm{R1}}{\mathrm{R1} + \mathrm{R2}} \right)$$

Example:

Given: *R1*=40_Ω, *R2*=10_Ω, V=100_V.

Solution: *V1*=80_V.

Current Divider (2, 4)



Equation:

$$I1 = I \cdot \left(\frac{R2}{R1 + R2} \right)$$

Example:

Given: $R1$=10_Ω, $R2$=6_Ω, I=15_A.

Solution: $I1$=5.6250_A.

Wire Resistance (2, 5)

Equation:

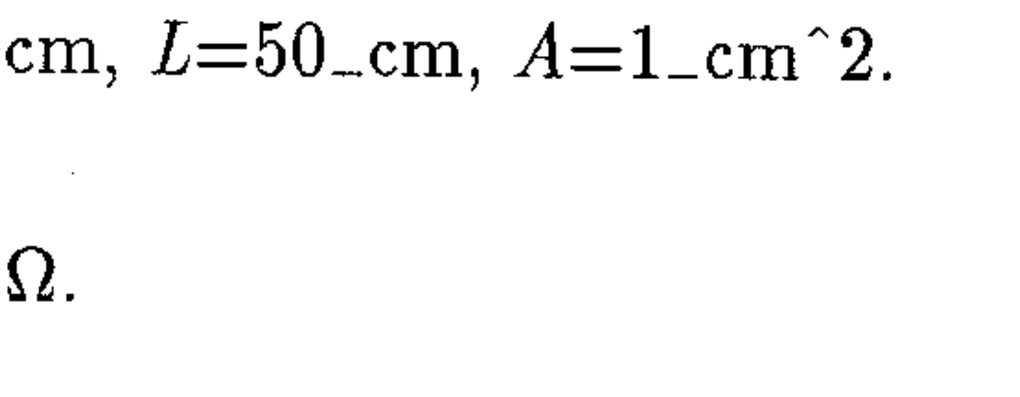
$$R = \frac{\rho \cdot L}{A}$$

Example:

Given: ρ=.0035_Ω∗cm, L=50_cm, A=1_cm^2.

Solution: R=0.175_Ω.

Series and Parallel R (2, 6)



Equations:

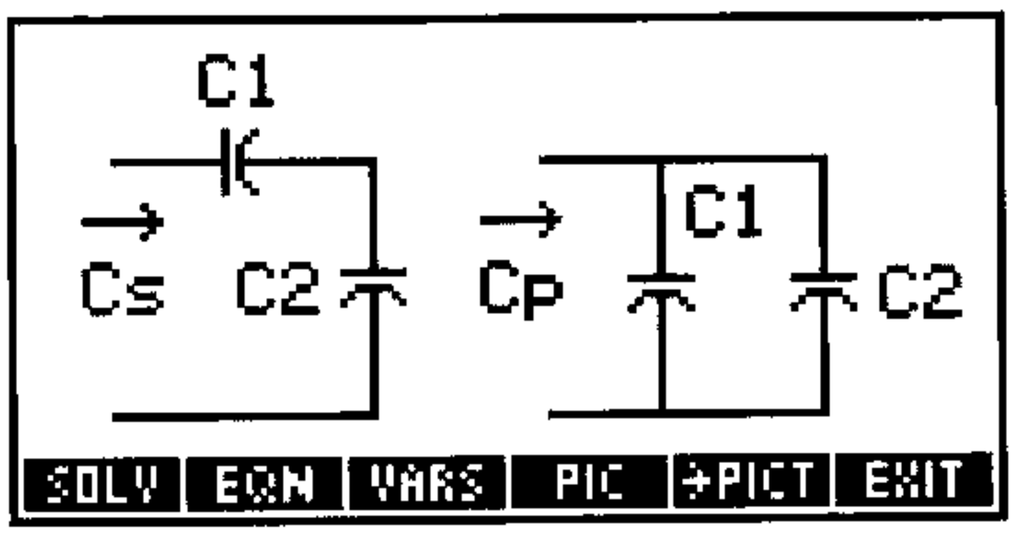
$$Rs = R1 + R2 \qquad \frac{1}{Rp} = \frac{1}{R1} + \frac{1}{R2}$$

Example:

Given: $R1{=}2_\Omega,\ R2{=}3_\Omega.$

Solution: $Rs{=}5_\Omega,\ Rp{=}1.2000_\Omega.$

Series and Parallel C (2, 7)



Equations:

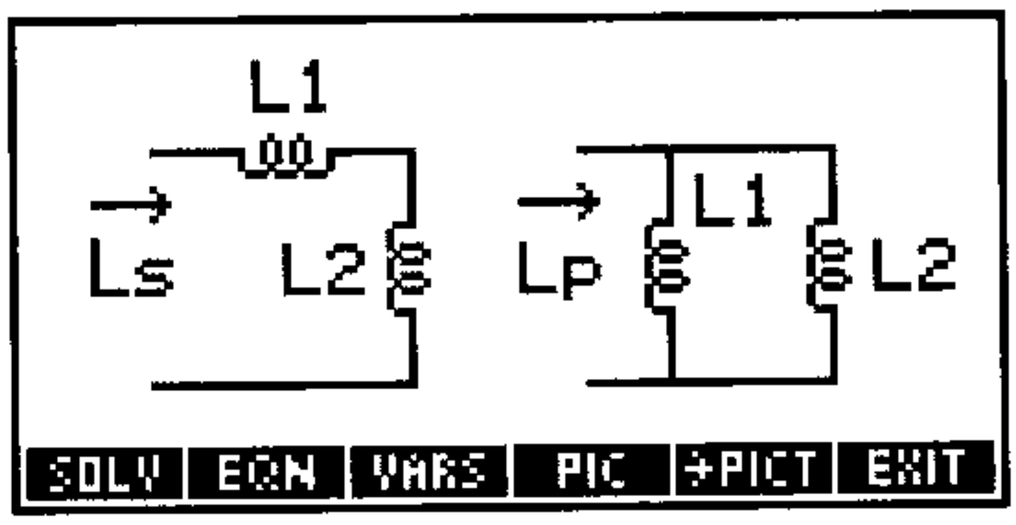
$$\frac{1}{\mathrm{Cs}} = \frac{1}{\mathrm{C1}} + \frac{1}{\mathrm{C2}} \qquad \mathrm{Cp} = \mathrm{C1} + \mathrm{C2}$$

Example:

Given: $C1{=}2_\mu\mathrm{F},\ C2{=}3_\mu\mathrm{F}.$

Solution: $Cs{=}1.2000_\mu\mathrm{F},\ Cp{=}5_\mu\mathrm{F}.$

Series and Parallel L (2, 8)



Equations:

$$Ls = L1 + L2 \qquad \frac{1}{Lp} = \frac{1}{L1} + \frac{1}{L2}$$

Example:

Given: $L1$=17_mH, $L2$=16.5_mH.

Solution: Ls=33.5000_mH, Lp=8.3731_mH.

Capacitive Energy (2, 9)

Equation:

$$E = \frac{C \cdot V^2}{2}$$

Example:

Given: E=.025_J, C=20_μF.

Solution: V=50_V.

Inductive Energy (2, 10)

Equation:

$$E = \frac{L \cdot I^2}{2}$$

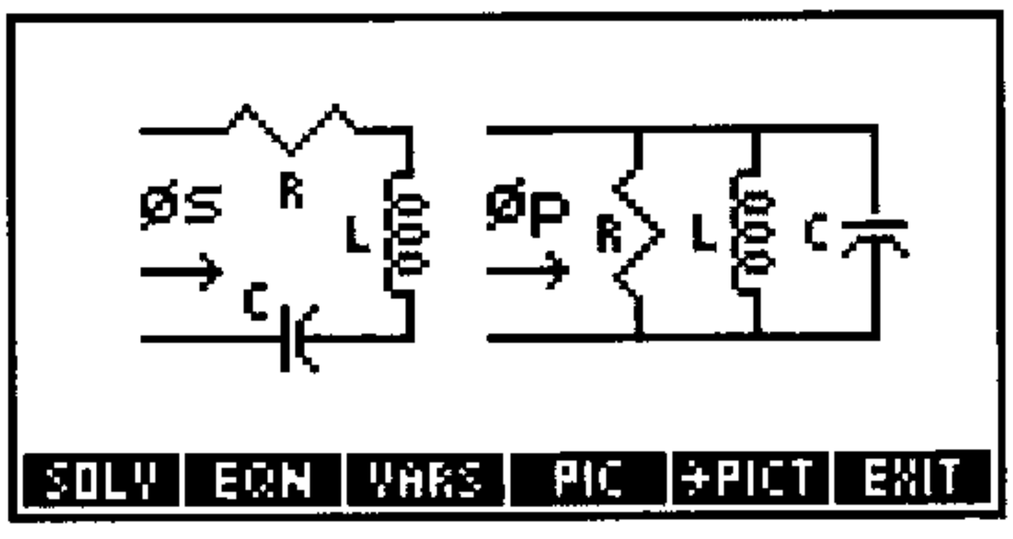
Example:

Given: E=4_J, L=15_mH.

Solution: I=23.0940_A.

RLC Current Delay (2, 11)

The phase delay (angle) is positive for current lagging voltage.



Equations:

$$\mathrm{TAN}\left(\phi \mathrm{s}\right) = \frac{\mathrm{XL} - \mathrm{XC}}{\mathrm{R}} \qquad \mathrm{TAN}\left(\phi \mathrm{p}\right) = \frac{\frac{1}{\mathrm{XC}} - \frac{1}{\mathrm{XL}}}{\frac{1}{\mathrm{R}}}$$

$$\mathrm{XC} = \frac{1}{\omega \cdot \mathrm{C}} \qquad \mathrm{XL} = \omega \cdot \mathrm{L} \qquad \omega = 2 \cdot \pi \cdot \mathrm{f}$$

Example:

Given: f=107_Hz, C=80_μF, L=20_mH, R=5_Ω.

Solution: ω=672.3008_r/s, ϕs=$-$45.8292_°, ϕp=$-$5.8772 _°, XC=18.5929_Ω, XL=13.4460_Ω.

DC Capacitor Current (2, 12)

These equations approximate the dc current required to change the voltage on a capacitor in a certain time interval.

Equations:

$$\mathrm{I} = \mathrm{C} \cdot \left(\frac{\Delta \mathrm{V}}{\Delta \mathrm{t}}\right) \qquad \Delta \mathrm{V} = \mathrm{Vf} - \mathrm{Vi} \qquad \Delta \mathrm{t} = \mathrm{tf} \cdot \mathrm{ti}$$

Example:

Given: C=15_μF, Vi=2.3_V, Vf=3.2_V, I=10_A, ti=0_s.

Solution: ΔV=.9000_V, Δt=1.3500_μs, tf=1.3500_μs.

Capacitor Charge (2, 13)

Equation:

$$\mathrm{q} = \mathrm{C} \cdot \mathrm{V}$$

Example:

Given: C=20_μF, V=100_V.

Solution: q=0.0020_C.

DC Inductor Voltage (2, 14)

These equations approximate the dc voltage induced in an inductor by a change in current in a certain time interval.

Equations:

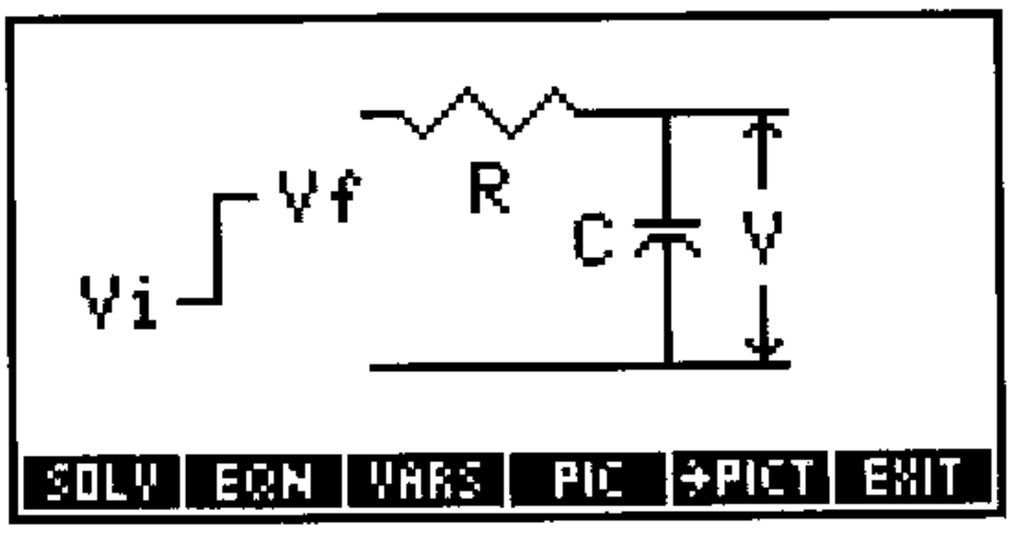
$$\mathrm{V} = -\mathrm{L} \cdot \left(\frac{\Delta \mathrm{I}}{\Delta \mathrm{t}} \right) \qquad \Delta \mathrm{I} = \mathrm{If} - \mathrm{Ii} \qquad \Delta \mathrm{t} = \mathrm{tf} - \mathrm{ti}$$

Example:

Given: L=100_mH, V=52_V, Δt=32_μs, Ii=23_A, ti=0_s.

Solution: ΔI=$-$0.0166_A, If=22.9834_A, tf=32_μs.

RC Transient (2, 15)



Equation:

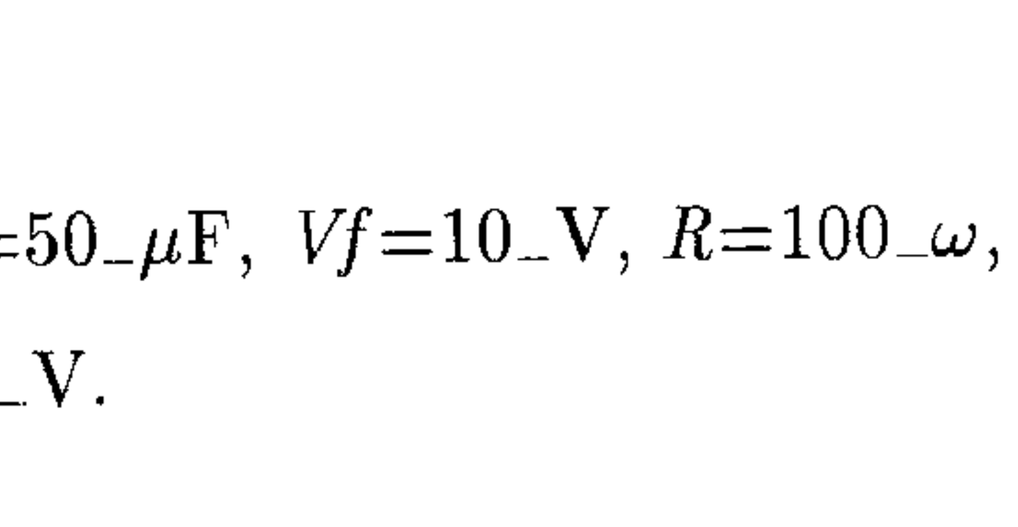
$$V = Vf - \left(Vf - Vi \right) \cdot e^{\frac{-t}{R \cdot C}}$$

Example:

Given: Vi=0_V, C=50_μF, Vf=10_V, R=100_ω, t=2_ms.

Solution: V=3.2968_V.

RL Transient (2, 16)



Equation:

$$I = \frac{1}{R} \cdot \left(Vf - \left(Vf - Vi \right) \cdot e^{\frac{-t \cdot R}{L}} \right)$$

Example:

Given: Vi=0_V, Vf=5_V, R=50_ω, L=50_mH, t=75_μs.

Solution: I=0.0072_A.

Resonant Frequency (2, 17)

Equations:

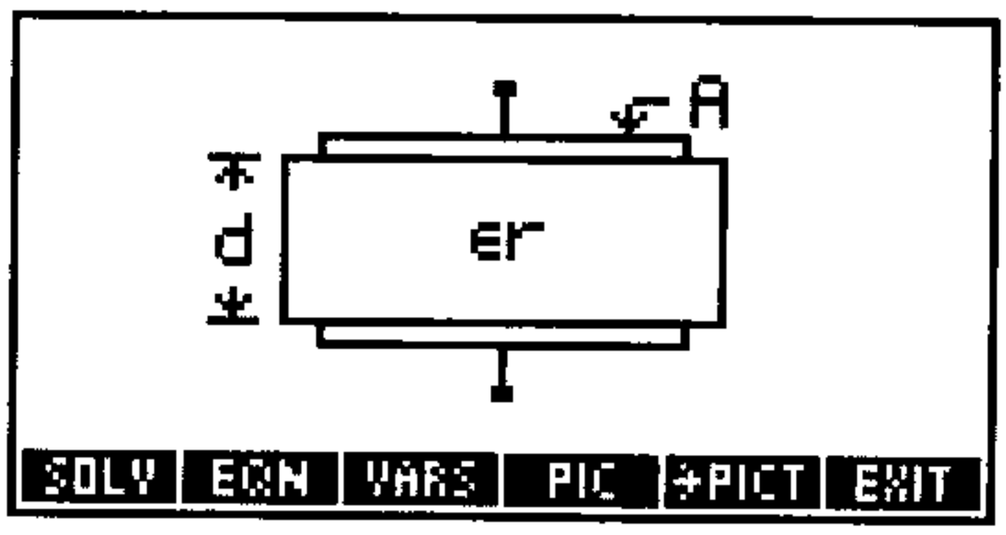
$$\omega 0 = \frac{1}{\sqrt{\mathrm{L}\cdot\mathrm{C}}} \qquad \mathrm{Qs} = \frac{1}{\mathrm{R}}\cdot\sqrt{\frac{\mathrm{L}}{\mathrm{C}}} \qquad \mathrm{Qp} = \mathrm{R}\cdot\sqrt{\frac{\mathrm{C}}{\mathrm{L}}} \qquad \omega 0 = 2\cdot\pi\cdot\mathrm{f0}$$

Example:

Given: L=500_mH, C=8_μF, R=10_ω.

Solution: $\omega 0$=500_r/s, Qs=25.0000, Qp=0.0400, $f0$=79.5775_Hz.

Plate Capacitor (2, 18)



Equation:

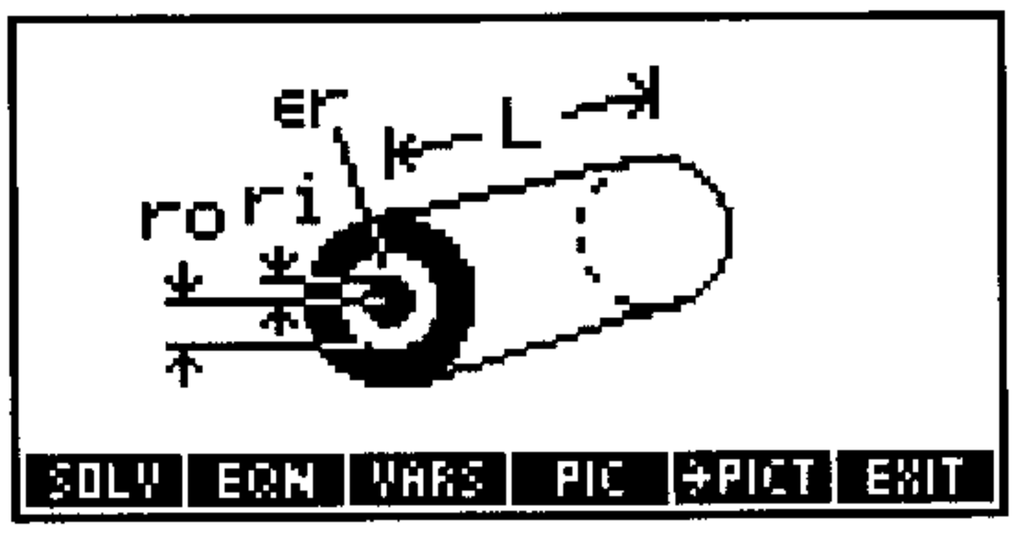
$$\mathrm{C} = \frac{\epsilon 0\cdot\epsilon\mathrm{r}\cdot\mathrm{A}}{\mathrm{d}}$$

Example:

Given: C=25_μF, ϵr=2.26, A=1_cm^2.

Solution: d=8.0042E−9_cm.

Cylindrical Capacitor (2, 19)



Equation:

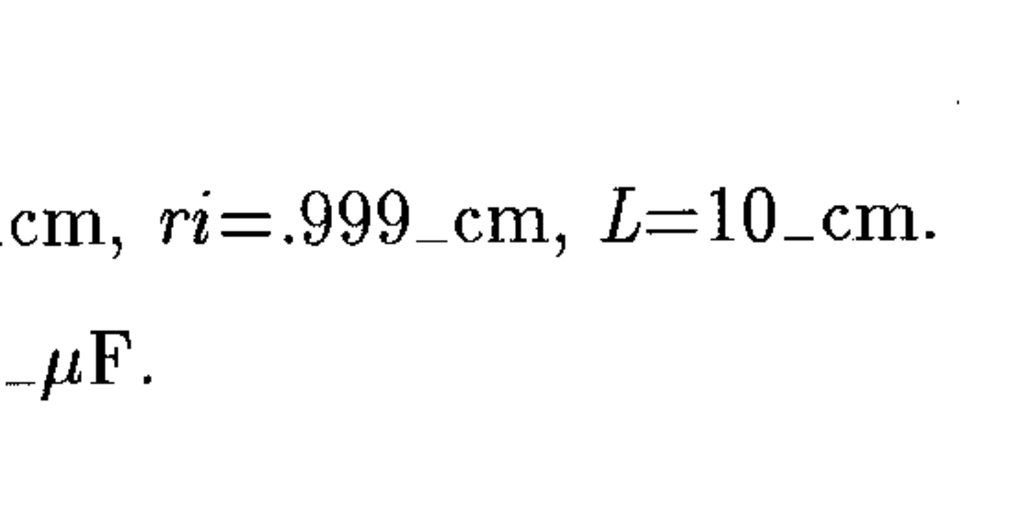
$$C = \frac{2 \cdot \pi \cdot \epsilon 0 \cdot \epsilon r \cdot L}{LN\left(\frac{ro}{ri}\right)}$$

Example:

Given: ϵr=1, ro=1_cm, ri=.999_cm, L=10_cm.

Solution: C=0.0056_μF.

Solenoid Inductance (2, 20)



Equation:

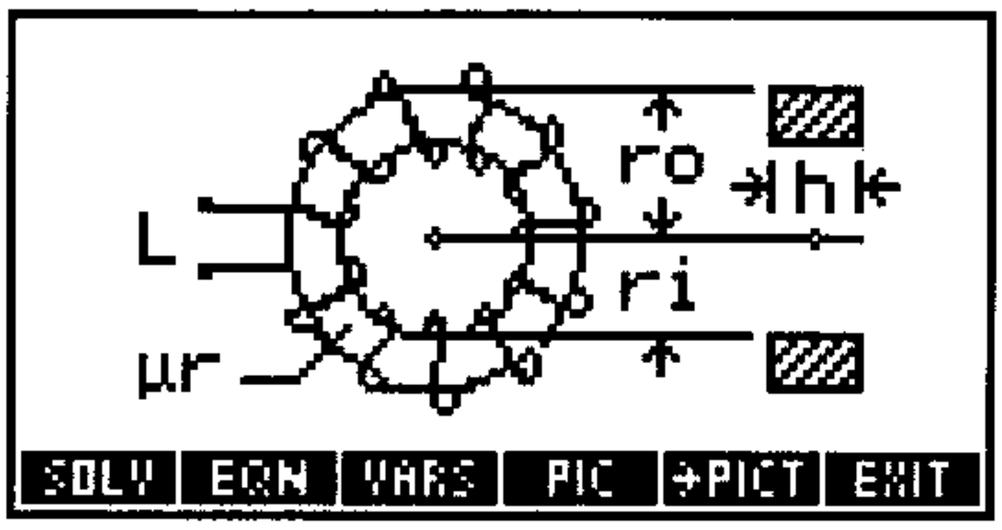
$$L = \mu 0 \cdot \mu r \cdot n^2 \cdot A \cdot h$$

Example:

Given: μr=2.5, n=40_1/cm, A=.2_cm^2, h=3_cm.

Solution: L=0.0302_mH.

Toroid Inductance (2, 21)



Equation:

$$L = \frac{\mu 0 \cdot \mu r \cdot N^2 \cdot h}{2 \cdot \pi} \cdot LN\left(\frac{ro}{ri}\right)$$

Example:

Given: μr=1, N=5000, h=2_cm, ri=2_cm, ro=4_cm.

Solution: L=69.3147_mH.

Sinusoidal Voltage (2, 22)

Equations:

$$V = Vmax \cdot SIN\left(\omega \cdot t + \phi\right) \qquad \omega = 2 \cdot \pi \cdot f$$

Example:

Given: $Vmax$=110_V, t=30_μs, f=60_Hz, ϕ=15_°.

Solution: ω=376.9911_r/s, V=29.6699_V.

Sinusoidal Current (2, 23)

Equations:

$$I = Imax \cdot SIN\left(\omega \cdot t + \phi\right) \qquad \omega = 2 \cdot \pi \cdot f$$

Example:

Given: t=32_s, $Imax$=10_A, ω=636_r/s, ϕ=30_°.

Solution: I=9.5983_A, f=101.2225_Hz.

Fluids (3)

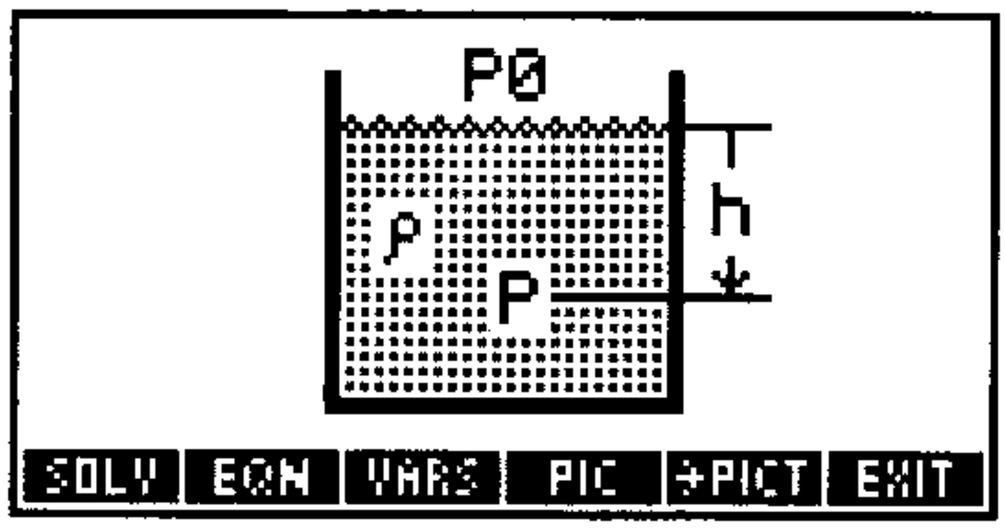
Variable Names and Descriptions

ϵ	Roughness
μ	Dynamic viscosity
ρ	Density
ΔP	Pressure change
Δy	Height change
ΣK	Total fitting coefficients
A	Cross-sectional area
$A1, A2$	Initial and final cross-sectional areas
D	Diameter
$D1, D2$	Initial and final diameters
h	Depth relative to $P0$ reference depth
hL	Head loss
L	Length
M	Mass flow rate
n	Kinematic viscosity
P	Pressure at h
$P0$	Reference pressure
$P1, P2$	Initial and final pressures
Q	Volume flow rate
Re	Reynolds number
$v1, v2$	Initial and final velocities
$vavg$	Average velocity
W	Power input
$y1, y2$	Initial and final heights

References: 3, 6, 9.

Pressure at Depth (3, 1)

This equation describes hydrostatic pressure for an incompressible fluid. Depth h is positive downward from the reference.



Equation:

$$\mathrm{P} = \mathrm{P0} + \rho \cdot \mathrm{g} \cdot \mathrm{h}$$

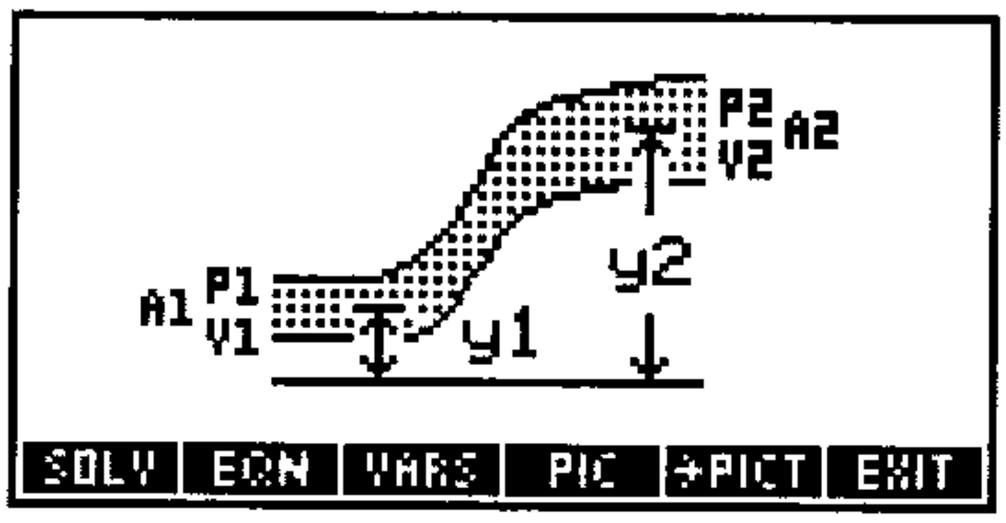
Example:

Given: h=100_m, ρ=1025.1817_kg/m^3, $P0$=1_atm.

Solution: P=1106.6848_kPa.

Bernoulli Equation (3, 2)

These equations represent the streamlined flow of an incompressible fluid.



Equations:

$$\frac{\Delta P}{\rho} + \frac{v2^2 - v1^2}{2} + g \cdot \Delta y = 0$$

$$\frac{\Delta P}{\rho} + \frac{v2^2 \cdot \left(1 - \left(\frac{A2}{A1}\right)^2\right)}{2} + g \cdot \Delta y = 0$$

$$\frac{\Delta P}{\rho} + \frac{v1^2 \cdot \left(\left(\frac{A1}{A2}\right)^2 - 1\right)}{2} + g \cdot \Delta y = 0$$

$$\Delta P = P2 - P1 \qquad \Delta y = y2 - y1 \qquad M = \rho \cdot Q$$

$$Q = A2 \cdot v2 \qquad Q = A1 \cdot v1$$

$$A1 = \frac{\pi \cdot D1^2}{4} \qquad A2 = \frac{\pi \cdot D2^2}{4}$$

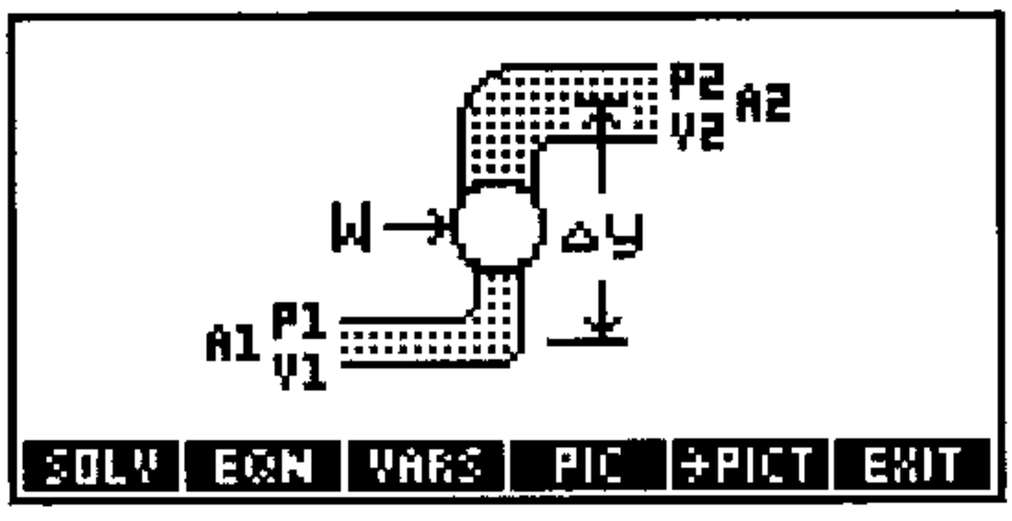
Example:

Given: $P2$=25_psi, $P1$=75_psi, $y2$=35_ft, $y1$=0_ft, $D1$=18_in, ρ=64_lb/ft^3, $v1$=100_ft/s.

Solution: Q=10602.8752_ft^3/min, M=678584.0132_lb/min, $v2$=122.4213_ft/s, $A2$=207.8633_in^2, $D2$=16.2684_in, $A1$=254.4690_in^2, ΔP=−50_psi, Δy=35_ft.

Flow with Losses (3, 3)

These equations extend Bernoulli's equation to include power input (or output) and head loss.



Equations:

$$M \cdot \left(\frac{\Delta P}{\rho} + \frac{v2^2 - v1^2}{2} + g \cdot \Delta y + hL \right) = W$$

$$M \cdot \left(\frac{\Delta P}{\rho} + \frac{v2^2 \cdot \left(1 - \left(\frac{A2}{A1} \right)^2 \right)}{2} + g \cdot \Delta y + hL \right) = W$$

$$M \cdot \left(\frac{\Delta P}{\rho} + \frac{v1^2 \cdot \left(\left(\frac{A1}{A2} \right)^2 - 1 \right)}{2} + g \cdot \Delta y + hL \right) = W$$

$$\Delta P = P2 - P1 \qquad \Delta y = y2 - y1 \qquad M = \rho \cdot Q$$

$$Q = A2 \cdot v2 \qquad Q = A1 \cdot v1$$

$$A1 = \frac{\pi \cdot D1^2}{4} \qquad A2 = \frac{\pi \cdot D2^2}{4}$$

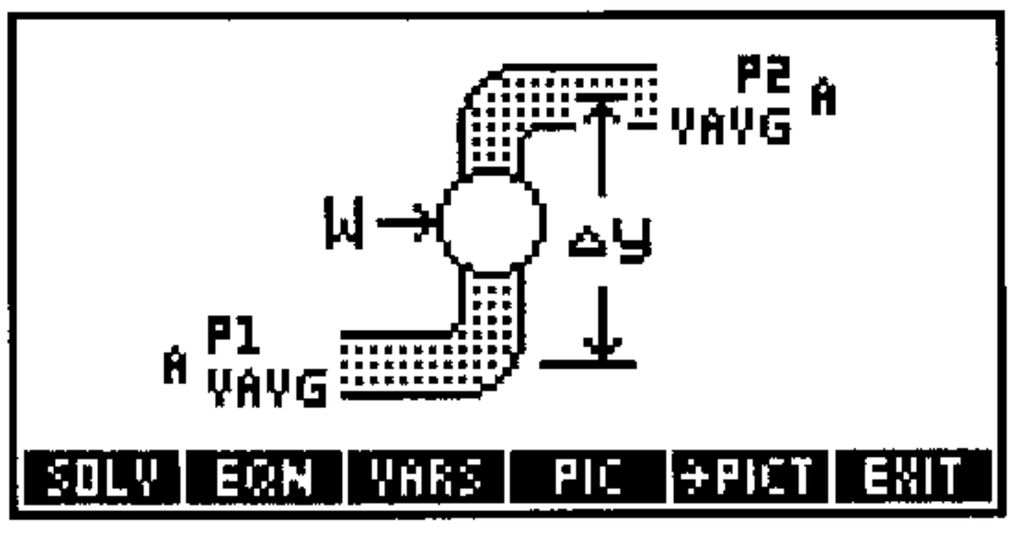
Example:

Given: $P2$=30_psi, $P1$=65_psi, $y2$=100_ft, $y1$=0_ft, ρ=64_lb/ft^3, $D1$=24_in, hL=2.0_ft^2/s^2, W=25_hp, $v1$=100_ft/s.

Solution: Q=18849.5559_ft^3/min, M=1206371.5790_lb/min, ΔP=−35_psi, Δy=100_ft, $v2$=93.1269_ft/s, $A1$=452.3893_in^2, $A2$=485.7773_in^2, $D2$=24.8699_in.

Flow in Full Pipes (3, 4)

These equations adapt Bernoulli's equation for flow in a round, full pipe, including power input (or output) and frictional losses. (See "FANNING" in chapter 3.)



Equations:

$$\rho \cdot \left(\frac{\pi \cdot D^2}{4} \right) \cdot vavg \cdot \left(\frac{\Delta P}{\rho} + g \cdot \Delta y + vavg^2 \cdot \left(2 \cdot f \cdot \left(\frac{L}{D} \right) + \frac{\Sigma K}{2} \right) \right) = W$$

$$\Delta P = P2 - P1 \qquad \Delta y = y2 - y1 \qquad M = \rho \cdot Q$$

$$Q = A \cdot vavg \qquad A = \frac{\pi \cdot D^2}{4} \qquad Re = \frac{D \cdot vavg \cdot \rho}{\mu} \qquad n = \frac{\mu}{\rho}$$

Example:

Given: ρ=62.4_lb/ft^3, D=12_in, $vavg$=8_ft/s, $P2$=15_psi, $P1$=20_psi, $y2$=40_ft, $y1$=0_ft, μ=.00002_lbf*s/ft^2, ΣK=2.25, ϵ=.02_in, L=250_ft.

Solution: ΔP=−5_psi, Δy=40_ft, A=113.0973_in^2, n=1.0312_ft^2/s, Q=376.9911_ft^3/min, M=23524.2458_lb/min, W=25.8897_hp, Re=775780.5.

Forces and Energy (4)

Variable Names and Descriptions

α	Angular acceleration
ω	Angular velocity
ωi,ωf	Initial and final angular velocities
ρ	Fluid density
τ	Torque
Θ	Angular displacement
a	Acceleration
A	Projected area relative to flow
ar	Centripetal acceleration at r
at	Tangential acceleration at r
Cd	Drag coefficient
E	Energy
F	Force at r or x, or Spring force (Hooke's Law), or Attractive force (Law of Gravitation), or Drag force (Drag Force)
I	Moment of inertia
k	Spring constant
Ki,Kf	Initial and final kinetic energies
m,$m1$,$m2$	Mass
N	Rotational speed
Ni,Nf	Initial and final rotational speeds
P	Instantaneous power
$Pavg$	Average power

Variable Names and Descriptions (continued)

r	Radius from rotation axis, or Separation distance (Law of Gravitation)
t	Time
v	Velocity
$vf, v1f, v2f$	Final velocity
$vi, v1i$	Initial velocity
W	Work
x	Displacement

Reference: 3.

Linear Mechanics (4, 1)

Equations:

$$F = m \cdot a \qquad Ki = \frac{1}{2} \cdot m \cdot vi^2 \qquad Kf = \frac{1}{2} \cdot m \cdot vf^2 \qquad W = F \cdot x$$

$$W = Kf - Ki \qquad P = F \cdot v \qquad Pavg = \frac{W}{t} \qquad vf = vi + a \cdot t$$

Example:

Given: t=10_s, m=50_lb, a=12.5_ft/s^2, vi=0_ft/s.

Solution: vf=125_ft/s, x=625_ft, F=19.4256_lbf, Ki=0_ft*lbf, Kf=12140.9961_ft*lbf, W=12140.9961_ft*lbf, $Pavg$=2.2075_hp.

Angular Mechanics (4, 2)

Equations:

$$\tau = \mathrm{I}\cdot\alpha \qquad \mathrm{Ki} = \frac{1}{2}\cdot\mathrm{I}\cdot\omega\mathrm{i}^2 \qquad \mathrm{Kf} = \frac{1}{2}\cdot\mathrm{I}\cdot\omega\mathrm{f}^2 \qquad \mathrm{W} = \tau\cdot\Theta$$

$$\mathrm{W} = \mathrm{Kf} - \mathrm{Ki} \qquad \mathrm{P} = \tau\cdot\omega \qquad \mathrm{Pavg} = \frac{\mathrm{W}}{\mathrm{t}} \qquad \omega\mathrm{f} = \omega\mathrm{i} + \alpha\cdot\mathrm{t}$$

$$\mathrm{at} = \alpha\cdot\mathrm{r} \qquad \omega = 2\cdot\pi\cdot\mathrm{N} \qquad \omega\mathrm{i} = 2\cdot\pi\cdot\mathrm{Ni} \qquad \omega\mathrm{f} = 2\cdot\pi\cdot\mathrm{Nf}$$

Example:

Given: I=1750_lb*in^2, Θ=360_°, r=3.5_in, α=10.5_r/min^2, ωi=0_r/s.

Solution: τ=1.1017E−3_ft*lbf, Ki=0_ft*lbf, W=6.9221E−3_ft*lbf, Kf=6.9221E−3_ft*lbf, at=8.5069E−4_ft/s^2, Ni=0_rpm, ωf=11.4868_r/min, t=1.0940_min, Nf=1.8282_rpm, $Pavg$=1.9174E−7_hp.

Centripetal Force (4, 3)

Equations:

$$\mathrm{F} = \mathrm{m}\cdot\omega^2\cdot\mathrm{r} \qquad \omega = \frac{\mathrm{v}}{\mathrm{r}} \qquad \mathrm{ar} = \frac{\mathrm{v}^2}{\mathrm{r}} \qquad \omega = 2\cdot\pi\cdot\mathrm{N}$$

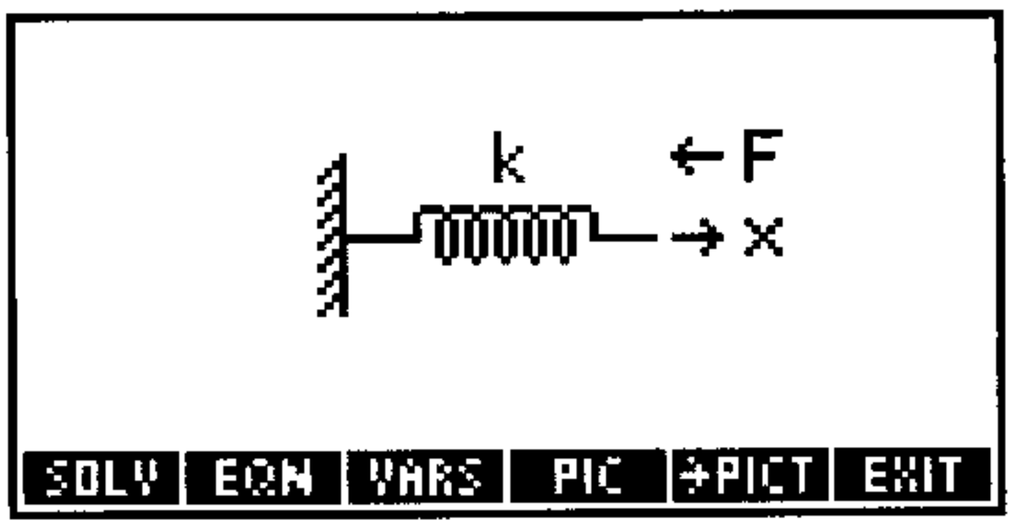
Example:

Given: m=1_kg, r=5_cm, N=2000_Hz.

Solution: ω=12566.3706_r/s, ar=7895683.5209_m/s, F=7895683.5209_N, v=628.3185_m/s.

Hooke's Law (4, 4)

The force is that exerted by the spring.



Equations:

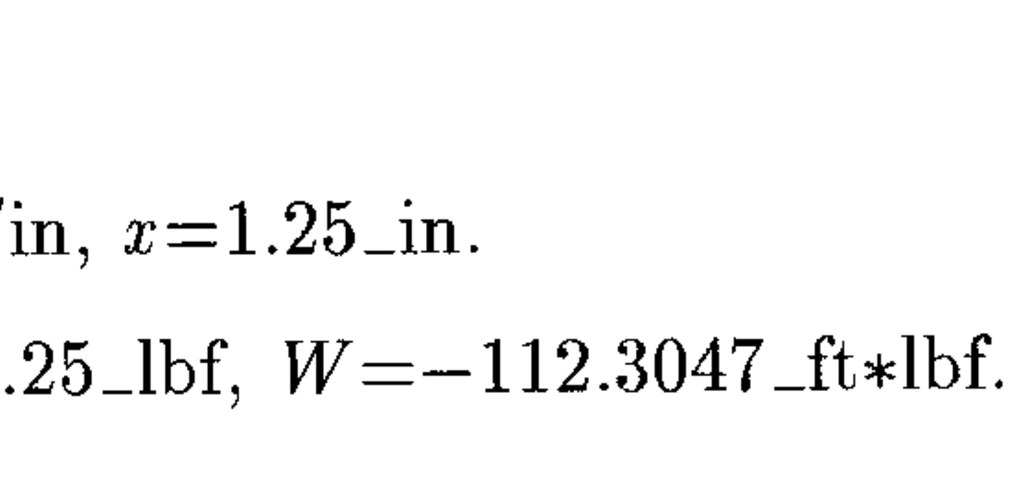
$$F = -k \cdot x \qquad W = \frac{-1}{2} \cdot k \cdot x^2$$

Example:

Given: k=1725_lbf/in, x=1.25_in.

Solution: F=−2156.25_lbf, W=−112.3047_ft∗lbf.

1D Elastic Collisions (4, 5)



Equations:

$$v1f = \frac{m1 - m2}{m1 + m2} \cdot v1i \qquad v2f = \frac{2 \cdot m1}{m1 + m2} \cdot v1i$$

Example:

Given: $m1$=10_kg, $m2$=25_kg, $v1i$=100_m/s.

Solution: $v1f$=−42.8571_m/s, $v2f$=57.1429_m/s.

Drag Force (4, 6)

Equation:

$$\mathrm{F} = \mathrm{Cd} \cdot \left(\frac{\rho \cdot \mathrm{v}^2}{2} \right) \cdot \mathrm{A}$$

Example:

Given: Cd=.05, ρ=1000_kg/m^3, A=7.5E6_cm^2, v=35_m/s.

Solution: F=22968750_N.

Law of Gravitation (4, 7)

Equation:

$$\mathrm{F} = \mathrm{G} \cdot \left(\frac{\mathrm{m1} \cdot \mathrm{m2}}{\mathrm{r}^2} \right)$$

Example:

Given: $m1$=2E15_kg, $m2$=2E18_kg, r=1000000_km.

Solution: F=266903.6_N.

Mass-Energy Relation (4, 8)

Equation:

$$\mathrm{E} = \mathrm{m} \cdot \mathrm{c}^2$$

Example:

Given: m=9.1E−31_kg.

Solution: E=8.1787E−14_J.

Gases (5)

Variable Names and Descriptions

λ	Mean free path
ρ	Flow density
$\rho 0$	Stagnation density
A	Flow area
At	Throat area
d	Molecular diameter
k	Specific heat ratio
M	Mach number
m	Mass
MW	Molecular weight
n	Number of moles, or Polytropic constant (Polytropic Processes)
P	Pressure, or Flow pressure (Isentropic Flow)
$P0$	Stagnation pressure
Pc	Pseudocritical pressure
Pi,Pf	Initial and final pressures
T	Temperature, or Flow temperature (Isentropic Flow)

Variable Names and Descriptions (continued)

$T0$	Stagnation temperature
Tc	Pseudocritical temperature
Ti, Tf	Initial and final temperatures
V	Volume
Vi, Vf	Initial and final volumes
$vrms$	Root-mean-square (rms) velocity
W	Work

References: 1, 3.

Ideal Gas Law (5, 1)

Equations:

$$\mathrm{P \cdot V = n \cdot R \cdot T} \qquad \mathrm{m = n \cdot MW}$$

Example:

Given: T=16.85_°C, P=1_atm, V=25_l, MW=36_g/gmol.

Solution: n=1.0506_gmol, m=3.7820E−2_kg.

Ideal Gas State Change (5, 2)

Equation:

$$\frac{\mathrm{Pf \cdot Vf}}{\mathrm{Tf}} = \frac{\mathrm{Pi \cdot Vi}}{\mathrm{Ti}}$$

Example:

Given: Pi=1.5_kPa, Pf=1.5_kPa, Vi=2_l, Ti=100_°C, Tf=373.15_K.

Solution: Vf=2_l.

Isothermal Expansion (5, 3)

These equations apply to an ideal gas.

Equations:

$$W = n \cdot R \cdot T \cdot LN\left(\frac{Vf}{Vi}\right) \qquad m = n \cdot MW$$

Example:

Given: Vi=2_l, Vf=125_l, T=300_°C, n=0.25_gmol, MW=64_g/gmol.

Solution: W=4926.4942_J, m=.016_kg.

Polytropic Processes (5, 4)

These equations describe a reversible pressure-volume change of an ideal gas such that $P * V^n$ is constant. Special cases include isothermal processes (n=1), isentropic processes (n=k, the specific heat ratio), and constant-pressure processes (n=0).

Equations:

$$\frac{Pf}{Pi} = \left(\frac{Vf}{Vi}\right)^{-n} \qquad \frac{Tf}{Ti} = \left(\frac{Pf}{Pi}\right)^{\frac{n-1}{n}}$$

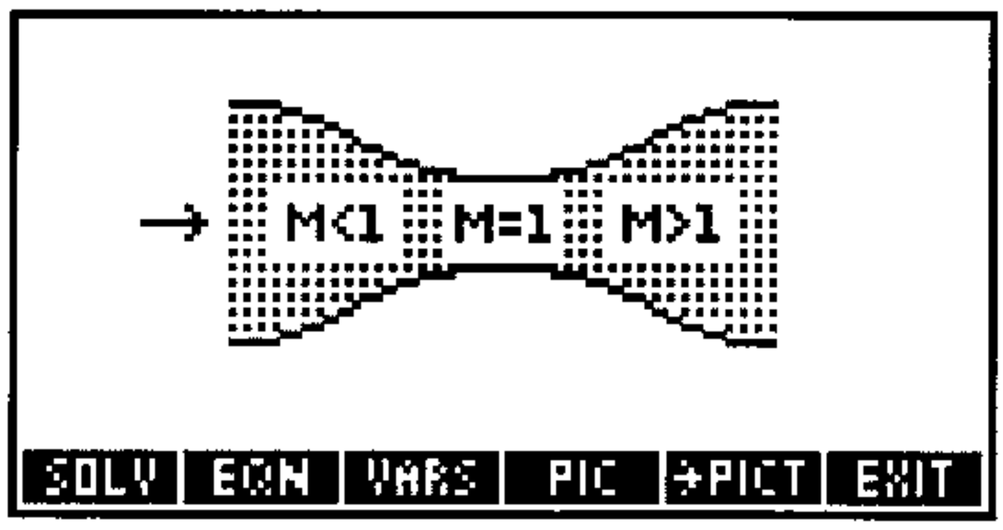
Example:

Given: Pi=15_psi, Pf=35_psi, Vi=1_ft^3, Vf=0.50_ft^3, Ti=75_°F.

Solution: n=1.2224, Tf=164.1117_°F.

Isentropic Flow (5, 5)

The calculation differs at velocities below and above Mach 1. The Mach number is based on the speed of sound in the compressible fluid.



Equations:

$$\frac{T}{T0} = \frac{2}{2 + (k-1)\cdot M^2} \qquad \frac{P}{P0} = \left(\frac{T}{T0}\right)^{\frac{k}{k-1}}$$

$$\frac{\rho}{\rho 0} = \left(\frac{T}{T0}\right)^{\frac{1}{k-1}}$$

$$\frac{A}{At} = \frac{1}{M}\cdot\left(\frac{2}{k+1}\cdot\left(1 + \frac{k-1}{2}\cdot M^2\right)\right)^{\frac{k+1}{2\cdot(k-1)}}$$

Example:

Given: k=2, M=.9, $T0$=26.85_°C, T=373.15_K, $\rho 0$=100_kg/m^3, $P0$=100_kPa, A=1_cm^2.

Solution: P=464.1152_kPa, At=0.9928_cm^2, ρ=215.4333_kg/m^3.

Real Gas Law (5, 6)

These equations adapt the ideal gas law to emulate real-gas behavior. (See "ZFACTOR" in chapter 3.)

Equations:

$$\mathrm{P}\cdot\mathrm{V} = \mathrm{n}\cdot\mathrm{Z}\cdot\mathrm{R}\cdot\mathrm{T} \qquad\qquad \mathrm{m} = \mathrm{n}\cdot\mathrm{MW}$$

Example:

Given: Pc=48_atm, Tc=298_K, P=5_kPa, V=10_l, MW=64_g/gmol, T=75_°C.

Solution: n=0.0173_gmol, m=1.1057E−3_kg.

Real Gas State Change (5, 7)

This equation adapts the ideal gas state-change equation to emulate real-gas behavior. (See "ZFACTOR" in chapter 3.)

Equation:

$$\frac{\mathrm{Pf}\cdot\mathrm{Vf}}{\mathrm{Zf}\cdot\mathrm{Tf}} = \frac{\mathrm{Pi}\cdot\mathrm{Vi}}{\mathrm{Zi}\cdot\mathrm{Ti}}$$

Example:

Given: Pc=48_atm, Pi=100_kPa, Pf=50_kPa, Ti=75_°C, Tc=298_K, Vi=10_l, Tf=250_°C.

(Remember Zf and Zi are automatically calculated by the HP 48 using these variables.)

Solution: Vf=30.1703_l.

Kinetic Theory (5, 8)

These equations describe properties of an ideal gas.

Equations:

$$P = \frac{n \cdot MW \cdot vrms^2}{3 \cdot V} \qquad vrms = \sqrt{\frac{3 \cdot R \cdot T}{MW}}$$

$$\lambda = \frac{1}{\sqrt{2} \cdot \pi \cdot \left(\frac{n \cdot NA}{V} \right) \cdot d^2} \qquad m = n \cdot MW$$

Example:

Given: P=100_kPa, V=2_l, T=26.85_°C, MW=18_g/gmol, d=2.5_nm.

Solution: $vrms$=644.7678_m/s, m=1.4433E−3_kg, n=.0802_gmol, λ=1.4916_nm.

Heat Transfer (6)

Variable Names and Descriptions

α	Expansion coefficient
δ	Elongation
$\lambda 1, \lambda 2$	Lower and upper wavelength limits
λmax	Wavelength of maximum emissive power
ΔT	Temperature difference
A	Area
c	Specific heat

Variable Names and Descriptions (continued)

$eb12$	Emissive power in the range $\lambda 1$ to $\lambda 2$
eb	Total emissive power
f	Fraction of emissive power in the range $\lambda 1$ to $\lambda 2$
$h, h1, h3$	Convective heat-transfer coefficient
$k, k1, k2, k3$	Thermal conductivity
$L, L1, L2, L3$	Length
m	Mass
Q	Heat capacity
q	Heat transfer rate
T	Temperature
Tc	Cold surface temperature (Conduction), or Cold fluid temperature
Th	Hot surface temperature, or Hot fluid temperature (Conduction + Convection)
Ti, Tf	Initial and final temperatures
U	Overall heat transfer coefficient

References: 7, 9.

Heat Capacity (6, 1)

Equations:

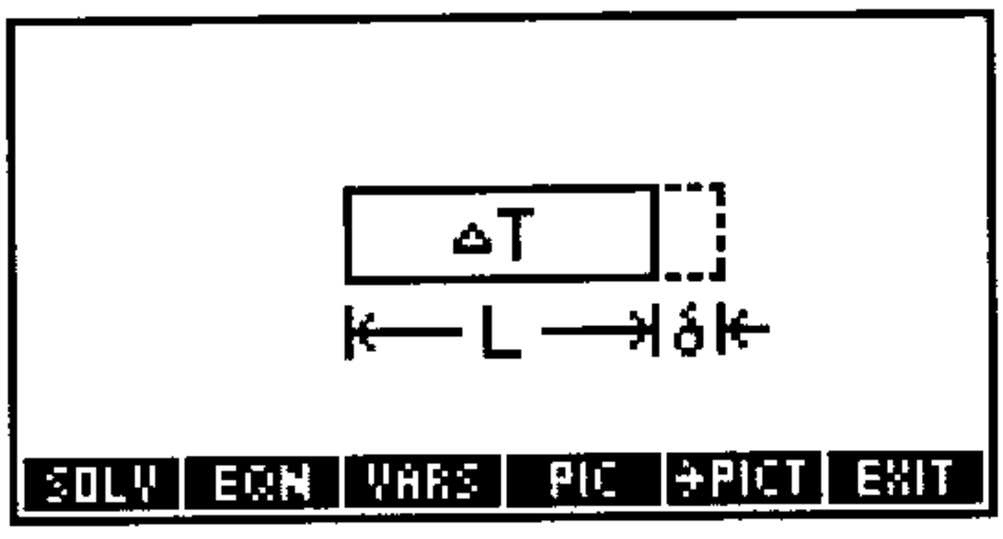
$$\mathrm{Q} = \mathrm{m}\cdot\mathrm{c}\cdot\Delta\mathrm{T} \qquad \mathrm{Q} = \mathrm{m}\cdot\mathrm{c}\cdot\left(\mathrm{Tf} - \mathrm{Ti}\right)$$

Example:

Given: ΔT=15_°C, Ti=0_°C, m=10_kg, Q=25_kJ.

Solution: Tf=15_°C, c=.1667_kJ/(kg∗K).

Thermal Expansion (6, 2)



Equations:

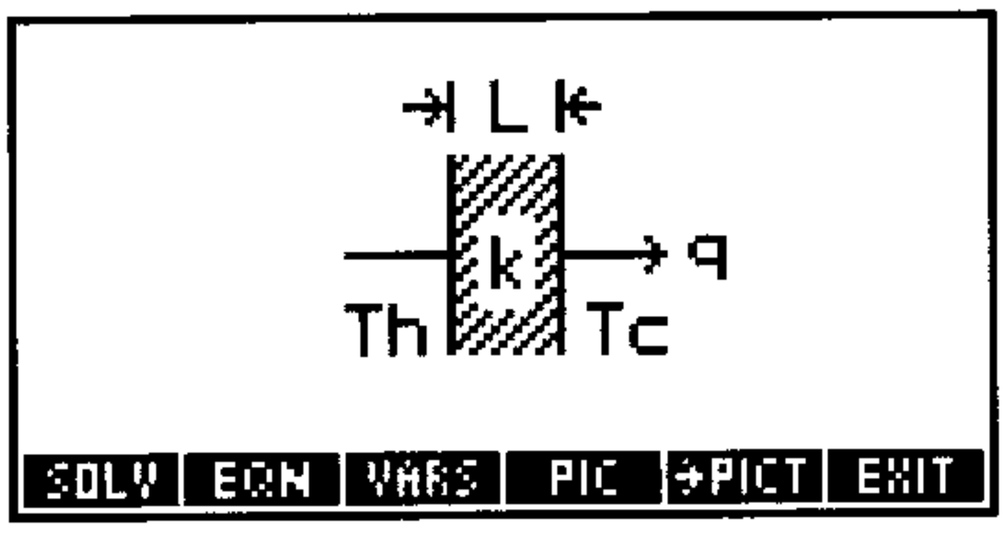
$$\delta = \alpha \cdot \mathrm{L} \cdot \Delta \mathrm{T} \qquad \delta = \alpha \cdot \mathrm{L} \cdot \left(\mathrm{Tf} - \mathrm{Ti} \right)$$

Example:

Given: ΔT=15_°C, L=10_m, Tf=25_°C, δ=1_cm.

Solution: Ti=10_°C, α=6.6667E−5_1/°C.

Conduction (6, 3)



Equations:

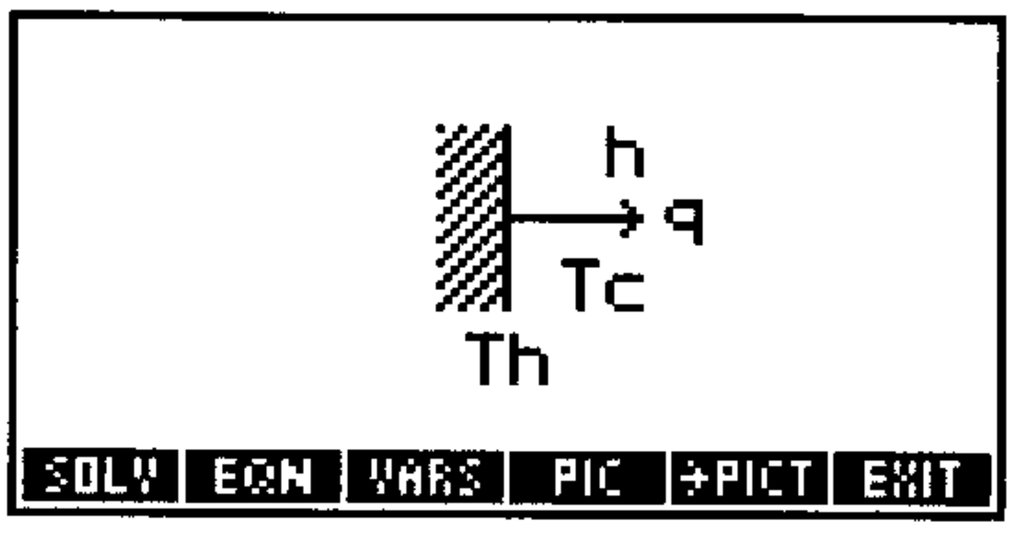
$$\mathrm{q} = \frac{\mathrm{k} \cdot \mathrm{A}}{\mathrm{L}} \cdot \Delta \mathrm{T} \qquad \mathrm{q} = \frac{\mathrm{k} \cdot \mathrm{A}}{\mathrm{L}} \cdot \left(\mathrm{Th} - \mathrm{Tc} \right)$$

Example:

Given: Tc=25_°C, Th=75_°C, A=12.5_m^2, L=1.5_cm, k=.12_W/(m*K).

Solution: q=5000_W, ΔT=50_°C.

Convection (6, 4)



Equations:

$$q = h \cdot A \cdot \Delta T \qquad q = h \cdot A \cdot \left(Th - Tc \right)$$

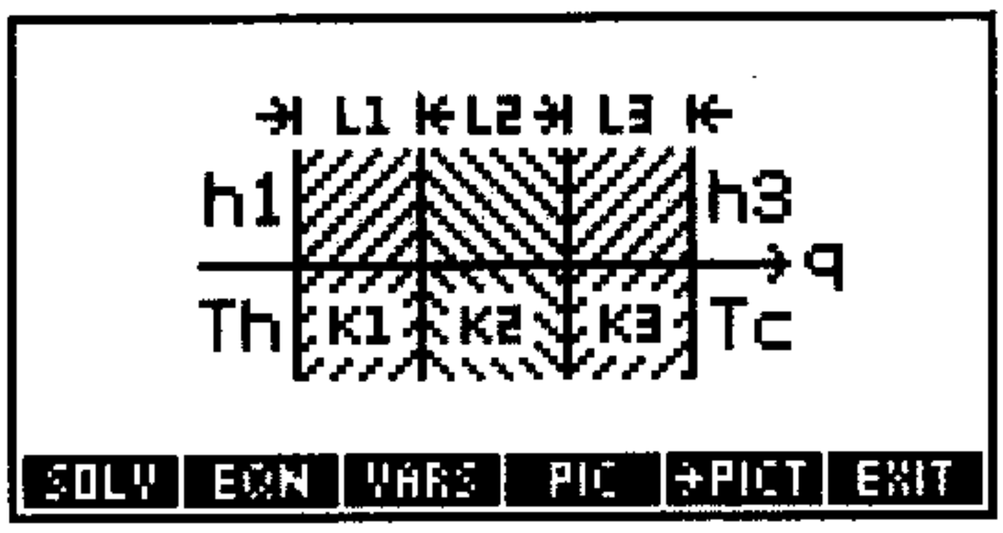
Example:

Given: Tc=300_K, A=200_m^2, h=.005_W/(m^2*K), q=10_W.

Solution: ΔT=10_°C, Th=36.8500_°C.

Conduction + Convection (6, 5)

If you have fewer than three layers, give the extra layers a zero thickness and any nonzero conductivity. The two temperatures are fluid temperatures—if instead you know a *surface* temperature, set the corresponding convective coefficient to 10^{499}.



Equations:

$$q = \frac{A \cdot \Delta T}{\frac{1}{h1} + \frac{L1}{k1} + \frac{L2}{k2} + \frac{L3}{k3} + \frac{1}{h3}} \qquad q = \frac{A \cdot \left(Th - Tc\right)}{\frac{1}{h1} + \frac{L1}{k1} + \frac{L2}{k2} + \frac{L3}{k3} + \frac{1}{h3}}$$

$$U = \frac{q}{A \cdot \Delta T} \qquad U = \frac{q}{A \cdot \left(Th - Tc\right)}$$

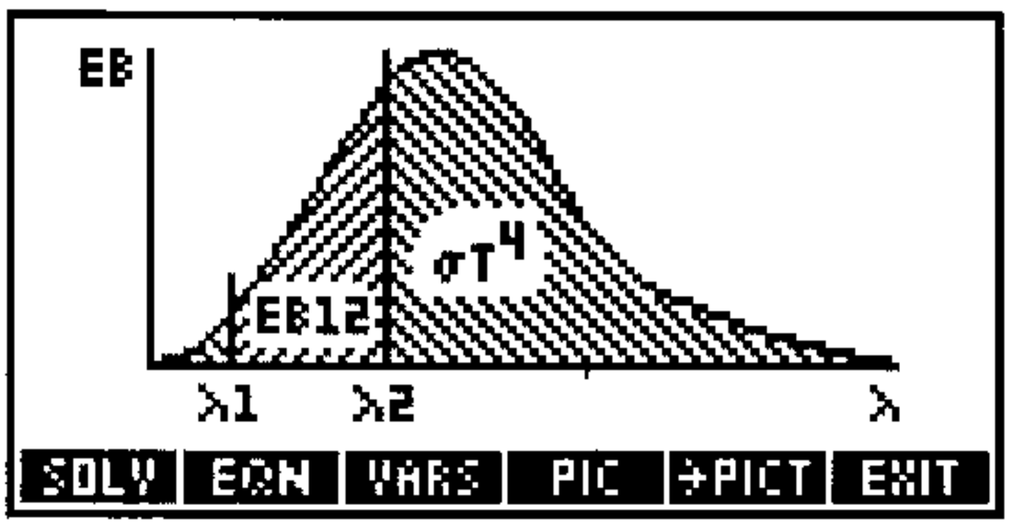
Example:

Given: ΔT=35_°C, Th=55_°C, A=10_m^2, $h1$=.05_W/(m^2*K), $h3$=.05_W/(m^2*K), $L1$=3_cm, $L2$=5_cm, $L3$=3_cm, $k1$=.1_W/(m*K), $k2$=.5_W/(m*K), $k3$=.1_W/(m*K).

Solution: Tc=20_°C, U=0.0246_W/(m^2*K), q=8.5995_W.

Black Body Radiation (6, 6)

See "F0λ" in chapter 3.



Equations:

$$\mathrm{eb} = \sigma \cdot \mathrm{T}^4 \qquad \mathrm{f} = \mathrm{F0}\lambda\left(\lambda 2; \mathrm{T}\right) - \mathrm{F0}\lambda\left(\lambda 1; \mathrm{T}\right)$$

$$\mathrm{eb12} = \mathrm{f} \cdot \mathrm{eb} \qquad \lambda\mathrm{max} \cdot \mathrm{T} = \mathrm{c3} \qquad \mathrm{q} = \mathrm{eb} \cdot \mathrm{A}$$

Example:

Given: T=1000_°C, $\lambda 1$=1000_nm, $\lambda 2$=600_nm, A=1_cm^2.

Solution: λmax=2276.0523_nm, eb=148984.2703_W/m^2, f=.0036, $eb12$=537.7264_W/m^2, q=14.8984_W.

Magnetism (7)

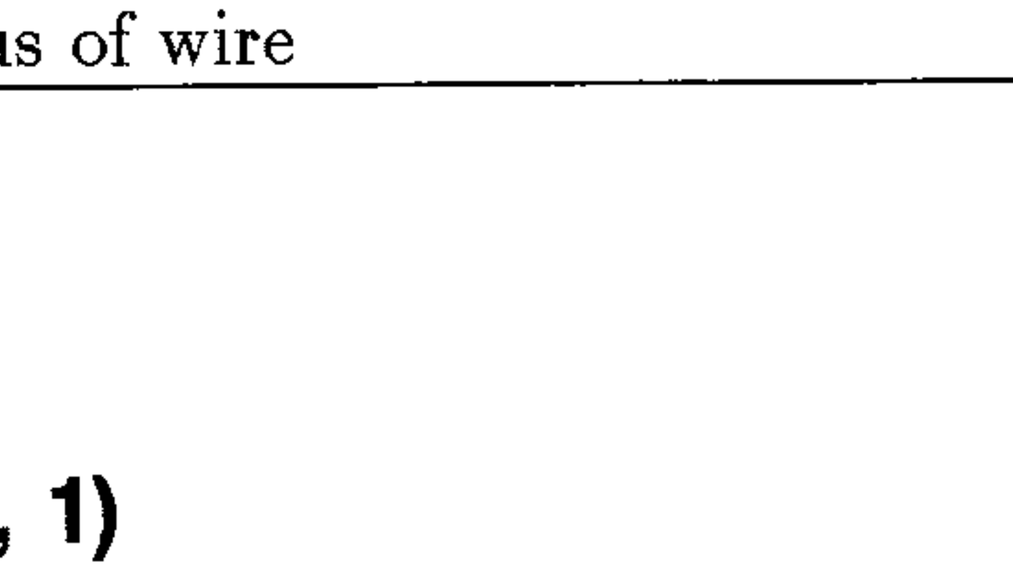
Variable Names and Descriptions

μr	Relative permeability
B	Magnetic field
d	Separation distance
Fba	Force
I, Ia, Ib	Current
L	Length
N	Total number of turns
n	Number of turns per unit length
r	Distance from center of wire
ri, ro	Inside and outside radii of toroid
rw	Radius of wire

Reference: 3.

Straight Wire (7, 1)

The magnetic field calculation differs depending upon whether the point is inside or outside the wire.



Equation:

$$B = \frac{\mu 0 \cdot \mu r \cdot I}{2 \cdot \pi \cdot r}$$

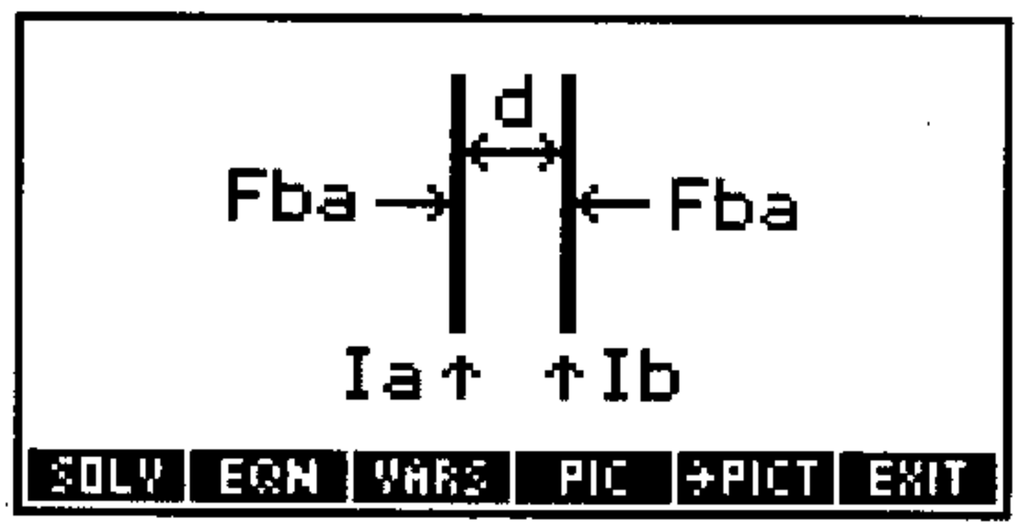
Example:

Given: μr=1, rw=.25_cm, r=.2_cm, I=25_A.

Solution: B=.0016_T.

Force between Wires (7, 2)

The force between wires is positive for an attractive force (for currents having the same sign).



Equation:

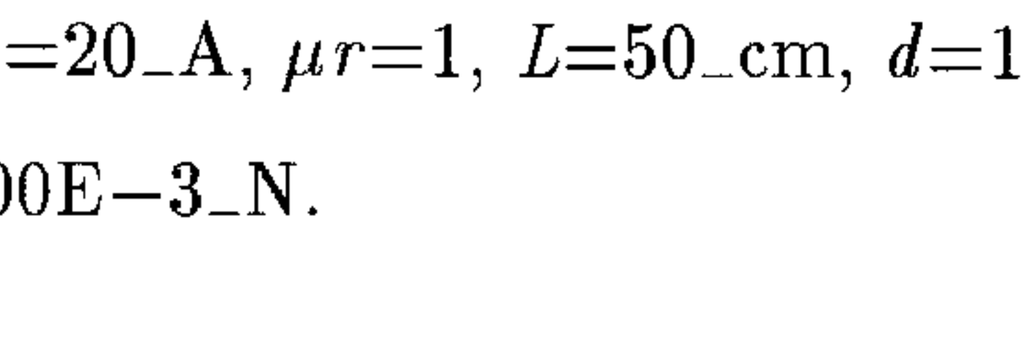
$$\mathrm{Fba} = \frac{\mu 0 \cdot \mu \mathrm{r} \cdot \mathrm{L} \cdot \mathrm{Ib} \cdot \mathrm{Ia}}{2 \cdot \pi \cdot \mathrm{d}}$$

Example:

Given: Ia=10_A, Ib=20_A, μr=1, L=50_cm, d=1_cm.

Solution: Fba=2.0000E−3_N.

Magnetic (B) Field in Solenoid (7, 3)



Equation:

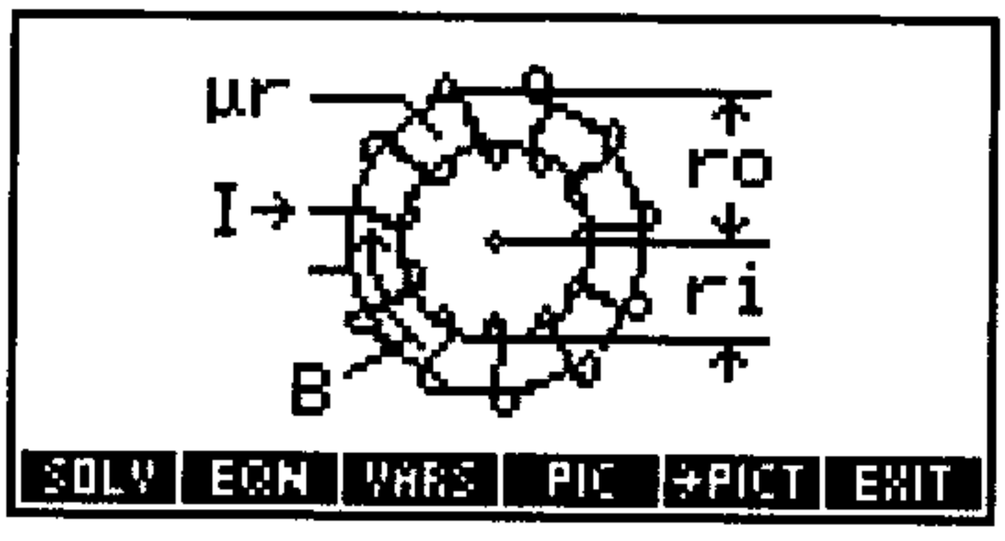
$$\mathrm{B} = \mu 0 \cdot \mu \mathrm{r} \cdot \mathrm{I} \cdot \mathrm{n}$$

Example:

Given: μr=10, n=50, I=1.25_A.

Solution: B=0.0785_T.

Magnetic (B) Field in Toroid (7, 4)



Equation:

$$\mathrm{B} = \frac{\mu 0 \cdot \mu \mathrm{r} \cdot \mathrm{I} \cdot \mathrm{N}}{2 \cdot \pi} \cdot \left(\frac{2}{\mathrm{ro} + \mathrm{ri}} \right)$$

Example:

Given: μr=10, N=50, ri=5_cm, ro=7_cm, I=10_A.

Solution: B=1.6667E−2_T.

Motion (8)

Variable Names and Descriptions

α	Angular acceleration
ω	Angular velocity (Circular Motion), or Angular velocity at t (Angular Motion)
$\omega 0$	Initial angular velocity
ρ	Fluid density
θ	Angular position at t
$\theta 0$	Initial angular position (Angular Motion), or Initial vertical angle (Projectile Motion)
a	Acceleration
A	Projected horizontal area
ar	Centripetal acceleration at r
Cd	Drag coefficient
m	Mass
M	Planet mass
N	Rotational speed
R	Horizontal range (Projectile Motion), or Planet radius (Escape Velocity)
r	Radius
t	Time
v	Velocity at t (Linear Motion), or Tangential velocity at r (Circular Motion), or Terminal velocity (Terminal Velocity), or Escape velocity (Escape Velocity)
$v0$	Initial velocity
vx	Horizontal component of velocity at t
vy	Vertical component of velocity at t
x	Horizontal position at t
$x0$	Initial horizontal position
y	Vertical position at t
$y0$	Initial vertical position

Reference: 3.

Linear Motion (8, 1)

Equations:

$$x = x0 + v0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 \qquad x = x0 + v \cdot t - \frac{1}{2} \cdot a \cdot t^2$$

$$x = x0 + \frac{1}{2} \cdot \left(v0 + v \right) \cdot t \qquad v = v0 + a \cdot t$$

Example:

Given: $x0$=0_m, x=100_m, t=10_s, $v0$=1_m/s.

Solution: v=19_m/s, a=1.8_m/s^2.

Object in Free Fall (8, 2)

Equations:

$$y = y0 + v0 \cdot t - \frac{1}{2} \cdot g \cdot t^2 \qquad y = y0 + v \cdot t + \frac{1}{2} \cdot g \cdot t^2$$

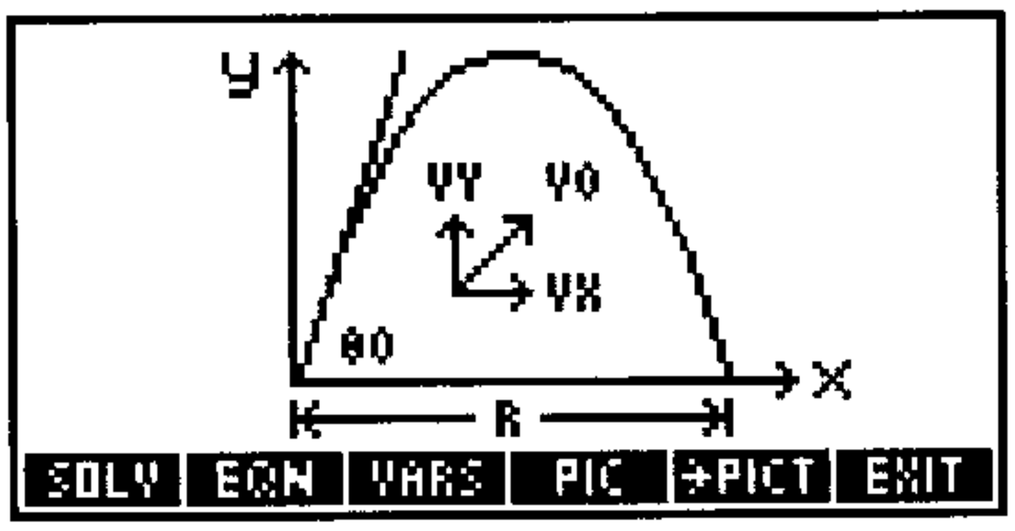
$$v^2 = v0^2 - 2 \cdot g \cdot \left(y - y0 \right) \qquad v = v0 - g \cdot t$$

Example:

Given: $y0$=1000_ft, y=0_ft, $v0$=0_ft/s.

Solution: t=7.8843_s, v=−253.6991_ft/s.

Projectile Motion (8, 3)



Equations:

$$\mathrm{x} = \mathrm{x0} + \mathrm{v0} \cdot \mathrm{COS}\left(\theta 0\right) \cdot \mathrm{t} \qquad \mathrm{y} = \mathrm{y0} + \mathrm{v0} \cdot \mathrm{SIN}\left(\theta 0\right) \cdot \mathrm{t} - \frac{1}{2} \cdot \mathrm{g} \cdot \mathrm{t}^2$$

$$\mathrm{vx} = \mathrm{v0} \cdot \mathrm{COS}\left(\theta 0\right) \qquad \mathrm{vy} = \mathrm{v0} \cdot \mathrm{SIN}\left(\theta 0\right) - \mathrm{g} \cdot \mathrm{t}$$

$$\mathrm{R} = \frac{\mathrm{v0}^2}{\mathrm{g}} \cdot \mathrm{SIN}\left(2 \cdot \theta 0\right)$$

Example:

Given: $x0$=0_ft, $y0$=0_ft, $\Theta 0$=45_°, $v0$=200_ft/s, t=10_s.

Solution: R=1243.2399_ft, vx=141.4214_ft/s, vy=−180.3186_ft/s, x=1414.2136_ft, y=−194.4864_ft.

Angular Motion (8, 4)

Equations:

$$\theta = \theta 0 + \omega 0 \cdot \mathrm{t} + \frac{1}{2} \cdot \alpha \cdot \mathrm{t}^2 \qquad \theta = \theta 0 + \omega \cdot \mathrm{t} - \frac{1}{2} \cdot \alpha \cdot \mathrm{t}^2$$

$$\theta = \theta 0 + \frac{1}{2} \cdot \left(\omega 0 + \omega\right) \cdot \mathrm{t} \qquad \omega = \omega 0 + \alpha \cdot \mathrm{t}$$

Example:

Given: $\Theta 0$=0_°, $\omega 0$=0_r/min, α=1.5_r/min^2, t=30_s.

Solution: Θ=10.7430_°, ω=.7500_r/min.

Circular Motion (8, 5)

Equations:

$$\omega = \frac{\mathrm{v}}{\mathrm{r}} \qquad \mathrm{ar} = \frac{\mathrm{v}^2}{\mathrm{r}} \qquad \omega = 2 \cdot \pi \cdot \mathrm{N}$$

Example:

Given: r=25_in, v=2500_ft/s.

Solution: ω=72000_r/min, ar=3000000_ft/s^2, N=11459.1559_rpm.

Terminal Velocity (8, 6)

Equation:

$$\mathrm{v} = \sqrt{\frac{2 \cdot \mathrm{m} \cdot \mathrm{g}}{\mathrm{Cd} \cdot \rho \cdot \mathrm{A}}}$$

Example:

Given: Cd=.15, ρ=.025lb/ft^3, A=100000_in^2, m=1250_lb.

Solution: v=1757.4709_ft/s.

Escape Velocity (8, 7)

Equation:

$$\mathrm{v} = \sqrt{\frac{2 \cdot \mathrm{G} \cdot \mathrm{M}}{\mathrm{R}}}$$

Example:

Given: M=1.5E23_lb, R=5000_mi.

Solution: v=3485.1106_ft/s.

Optics (9)

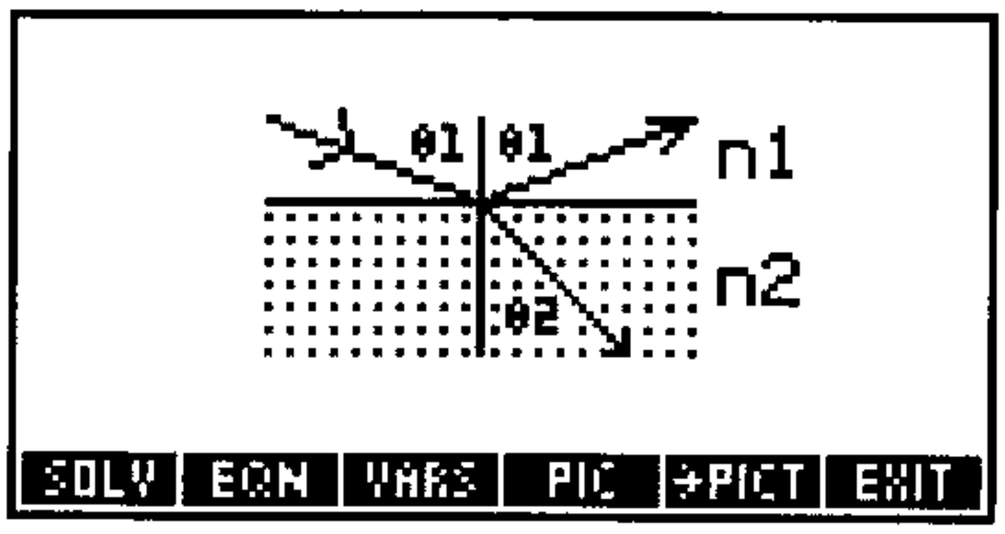
Variable Names and Descriptions

$\theta 1$	Angle of incidence
$\theta 2$	Angle of refraction
θB	Brewster angle
θc	Critical angle
f	Focal length
m	Magnification
$n, n1, n2$	Index of refraction
$r, r1, r2$	Radius of curvature
u	Distance to object
v	Distance to image

For reflection and refraction problems, the focal length and radius of curvature are positive in the direction of the outgoing light (reflected or refracted). The object distance is positive in front of the surface. The image distance is positive in the direction of the outgoing light (reflected or refracted). The magnification is positive for an upright image.

Reference: 3.

Law of Refraction (9, 1)



Equation:

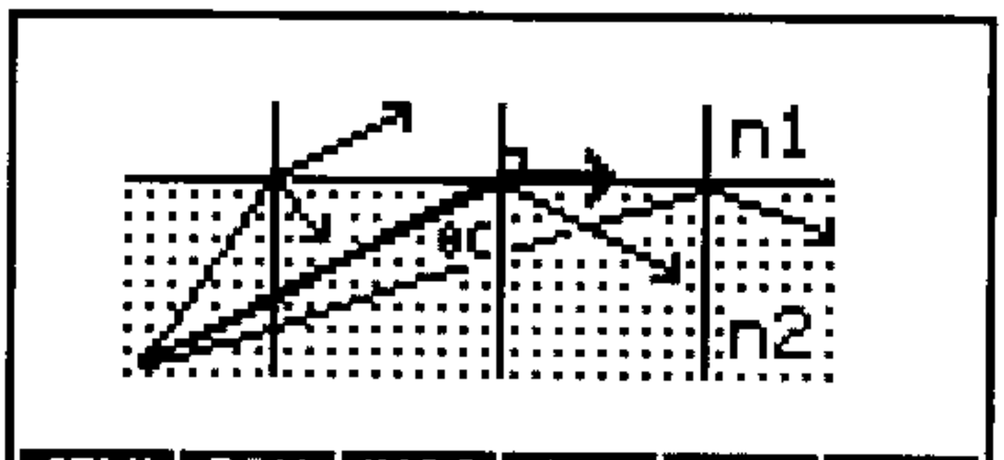
$$\mathrm{n1}\cdot\mathrm{SIN}\left(\theta 1\right) = \mathrm{n2}\cdot\mathrm{SIN}\left(\theta 2\right)$$

Example:

Given: $n1=1$, $n2=1.333$, $\theta 1=45_°$.

Solution: $\theta 2=32.0367_°$.

Critical Angle (9, 2)



Equation:

$$\mathrm{SIN}\left(\theta\mathrm{c}\right) = \frac{\mathrm{n1}}{\mathrm{n2}}$$

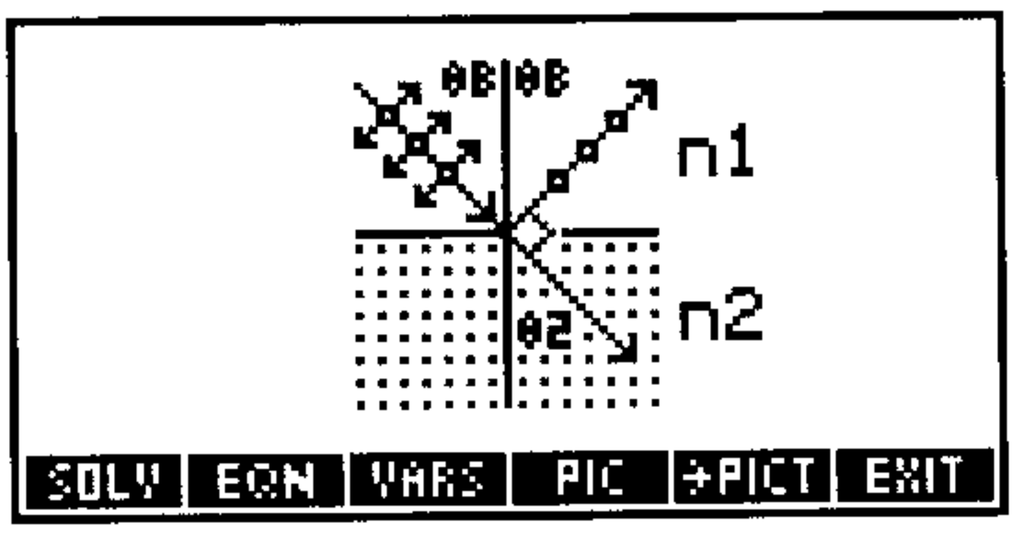
Example:

Given: $n1{=}1,\ n2{=}1.5.$

Solution: $\theta c{=}41.8103_°.$

Brewster's Law (9, 3)

The Brewster angle is the angle of incidence at which the reflected wave is completely polarized.



Equations:

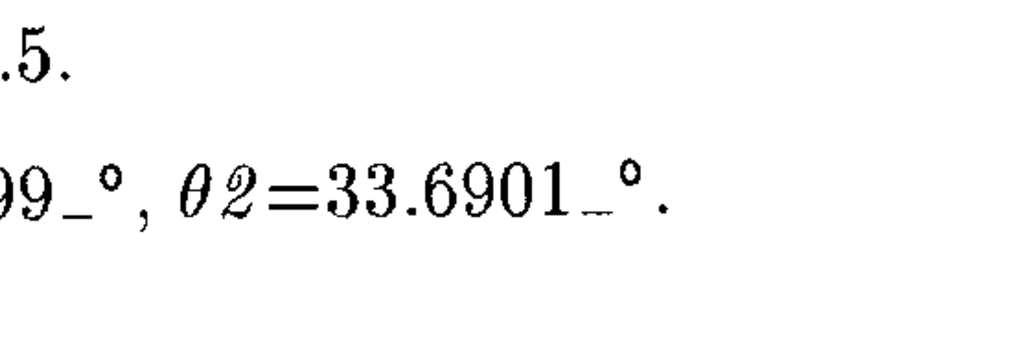
$$\mathrm{TAN}\left(\theta\mathrm{B}\right) = \frac{\mathrm{n2}}{\mathrm{n1}} \qquad \theta\mathrm{B} + \theta 2 = 90$$

Example:

Given: $n1{=}1,\ n2{=}1.5.$

Solution: $\theta B{=}56.3099_°,\ \theta 2{=}33.6901_°.$

Spherical Reflection (9, 4)



Equations:

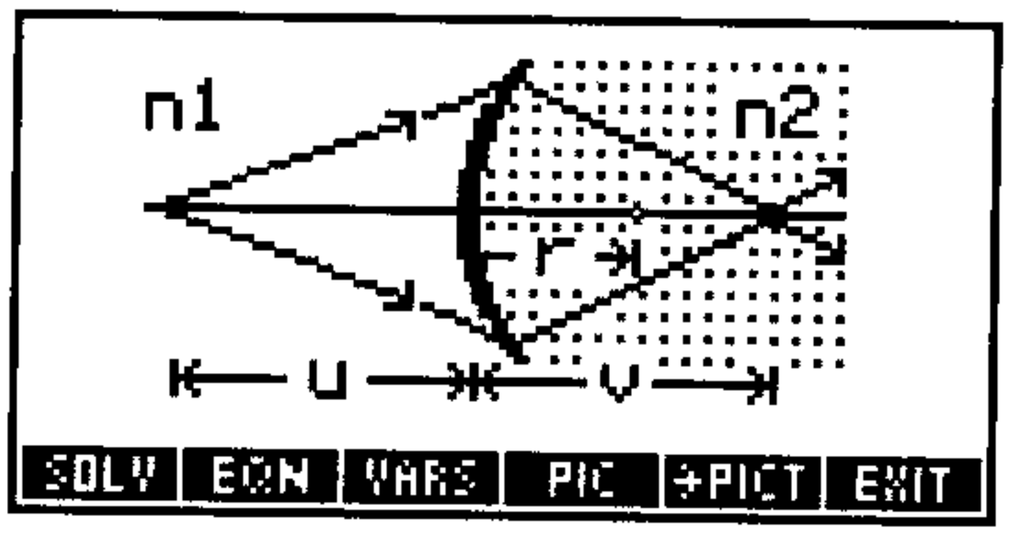
$$\frac{1}{u}+\frac{1}{v}=\frac{1}{f} \qquad f=\frac{1}{2}\cdot r \qquad m=\frac{-v}{u}$$

Example:

Given: u=10_cm, v=300_cm, r=19.35_cm.

Solution: m=−30, f=9.6774_cm.

Spherical Refraction (9, 5)



Equation:

$$\frac{n1}{u}+\frac{n2}{v}=\frac{n2-n1}{r}$$

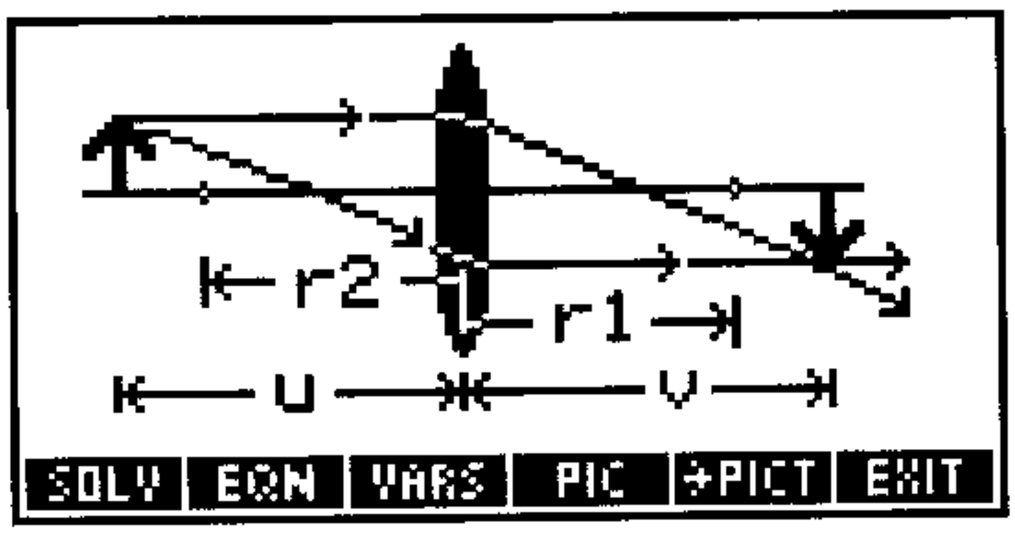
Example:

Given: u=8_cm, v=12_cm, r=2_cm, $n1$=1.

Solution: $n2$=1.5000.

Thin Lens (9, 6)

r1 is for the front surface, and *r2* is for the back surface.



Equations:

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f} \qquad \frac{1}{f} = \left(n - 1\right) \cdot \left(\frac{1}{r1} - \frac{1}{r2}\right) \qquad m = \frac{-v}{u}$$

Example:

Given: $r1 = 5_\text{cm}$, $r2 = 20_\text{cm}$, $n = 1.5$, $u = 50_\text{cm}$.

Solution: $f = 13.3333_\text{cm}$, $v = 18.1818_\text{cm}$, $m = -.3636$.

Oscillations (10)

Variable Names and Descriptions

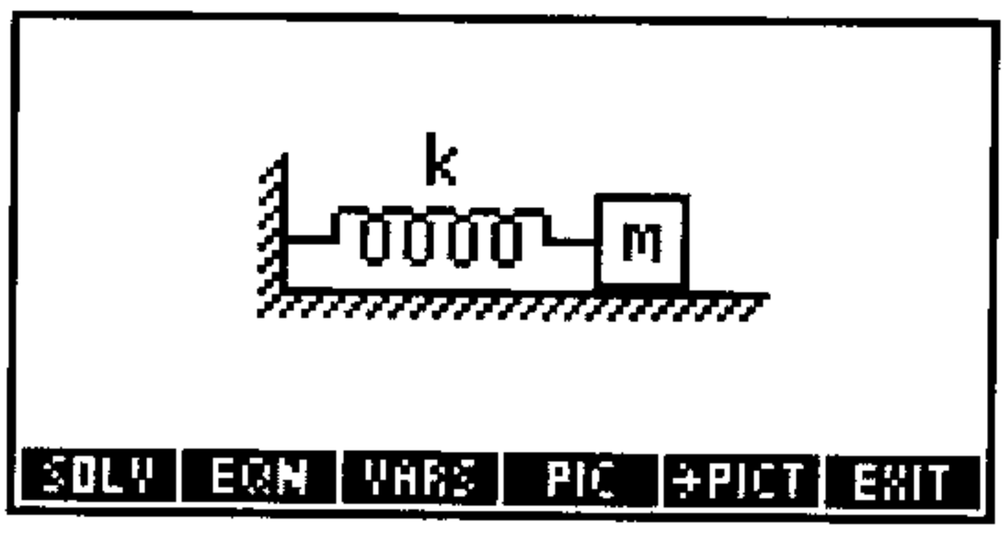
ω	Angular frequency
ϕ	Phase angle
θ	Cone angle
a	Acceleration at t
f	Frequency
G	Shear modulus of elasticity
h	Cone height

Variable Names and Descriptions (continued)

I	Moment of inertia
J	Polar moment of inertia
k	Spring constant
L	Length of pendulum
m	Mass
t	Time
T	Period
v	Velocity at t
x	Displacement at t
xm	Displacement amplitude

Reference: 3.

Mass-Spring System (10, 1)



Equations:

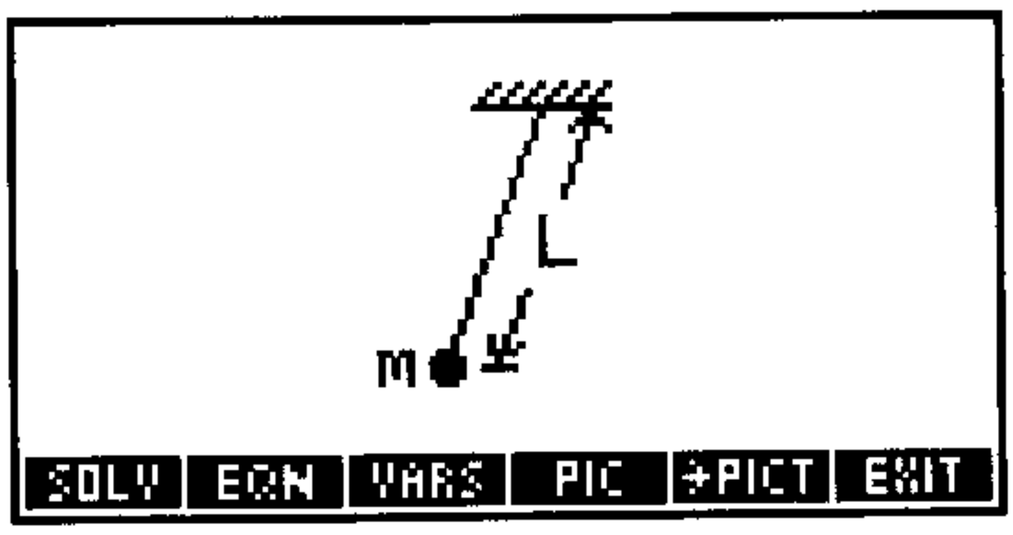
$$\omega = \sqrt{\frac{\mathrm{k}}{\mathrm{m}}} \qquad \mathrm{T} = \frac{2 \cdot \pi}{\omega} \qquad \omega = 2 \cdot \pi \cdot \mathrm{f}$$

Example:

Given: k=20_N/m, m=5_kg.

Solution: ω=2_r/s, T=3.1416_s, f=.3183_Hz.

Simple Pendulum (10, 2)



Equations:

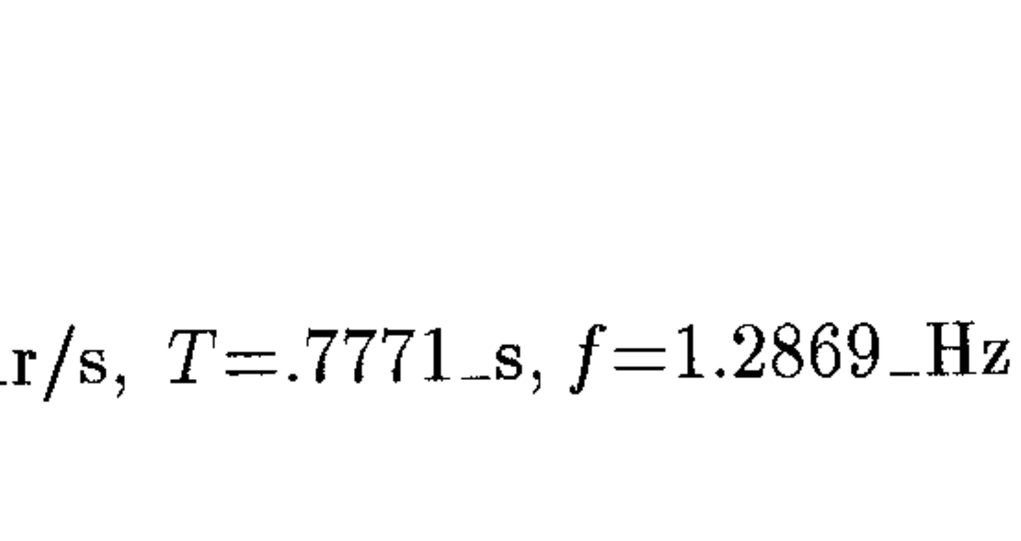
$$\omega = \sqrt{\frac{g}{L}} \qquad T = \frac{2 \cdot \pi}{\omega} \qquad \omega = 2 \cdot \pi \cdot f$$

Example:

Given: L=15_cm.

Solution: ω=8.0856_r/s, T=.7771_s, f=1.2869_Hz.

Conical Pendulum (10, 3)



Equations:

$$\omega = \sqrt{\frac{g}{h}} \qquad h = L \cdot \mathrm{COS}\left(\theta\right) \qquad T = \frac{2 \cdot \pi}{\omega}$$

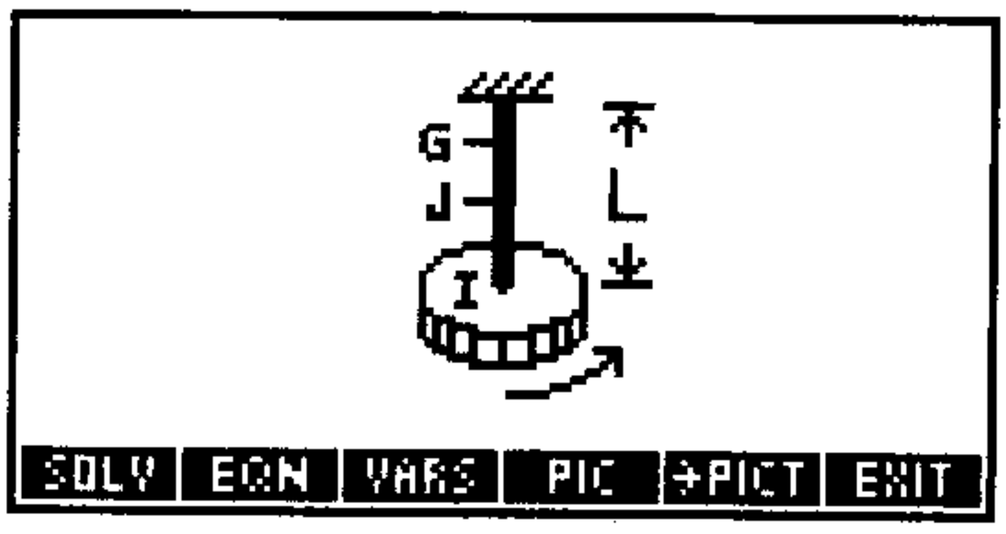
$$\omega = 2 \cdot \pi \cdot f$$

Example:

Given: L=25_cm, h=20_cm.

Solution: θ=36.899_°, T=.8973_s, ω=7.0024_r/s, f=1.1145_Hz.

Torsional Pendulum (10, 4)



Equations:

$$\omega = \sqrt{\frac{\mathrm{G}\cdot\mathrm{J}}{\mathrm{L}\cdot\mathrm{I}}} \qquad \mathrm{T} = \frac{2\cdot\pi}{\omega} \qquad \omega = 2\cdot\pi\cdot\mathrm{f}$$

Example:

Given: G=1000_kPa, J=17_mm^4, L=26_cm, I=50_kg*m^2.

Solution: ω=1.1435E−3_r/s, f=1.8200E−4_Hz, T=5494.4862_s.

Simple Harmonic (10, 5)

Equations:

$$\mathrm{x} = \mathrm{xm}\cdot\mathrm{COS}\left(\omega\cdot\mathrm{t}+\phi\right) \qquad \mathrm{v} = -\omega\cdot\mathrm{xm}\cdot\mathrm{SIN}\left(\omega\cdot\mathrm{t}+\phi\right)$$

$$\mathrm{a} = -\omega^2\cdot\mathrm{xm}\cdot\mathrm{COS}\left(\omega\cdot\mathrm{t}+\phi\right) \qquad \omega = 2\cdot\pi\cdot\mathrm{f}$$

Example:

Given: xm=10_cm, ω=15_r/s, ϕ=25_°, t=25_μs.

Solution: x=9.0615_cm, v=−0.6344_m/s, a=−20.3884_m/s^2, f=2.3873_Hz.

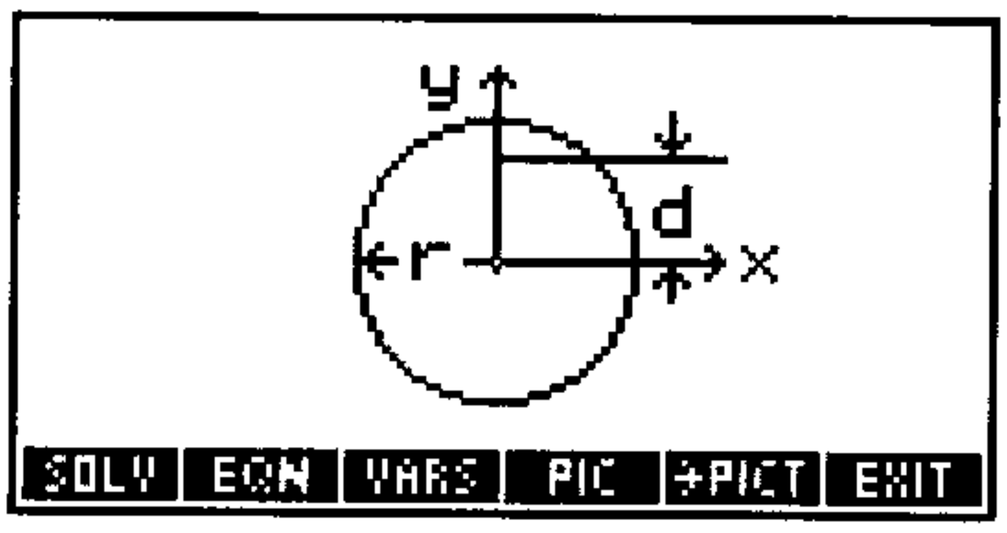
Plane Geometry (11)

Variable Names and Descriptions

β	Central angle of polygon
θ	Vertex angle of polygon
A	Area
b	Base length (Rectangle, Triangle), or Length of semiaxis in x direction (Ellipse)
C	Circumference
d	Distance to rotation axis in y direction
h	Height (Rectangle, Triangle), or Length of semiaxis in y direction (Ellipse)
I,Ix	Moment of inertia about x axis
Id	Moment of inertia in x direction at d
Iy	Moment of inertia about y axis
J	Polar moment of inertia at centroid
L	Side length of polygon
n	Number of sides
P	Perimeter
r	Radius
ri,ro	Inside and outside radii
rs	Distance to side of polygon
rv	Distance to vertex of polygon
v	Horizontal distance to vertex

Reference: 4.

Circle (11, 1)



Equations:

$$A = \pi \cdot r^2 \qquad C = 2 \cdot \pi \cdot r \qquad I = \frac{\pi \cdot r^4}{4}$$

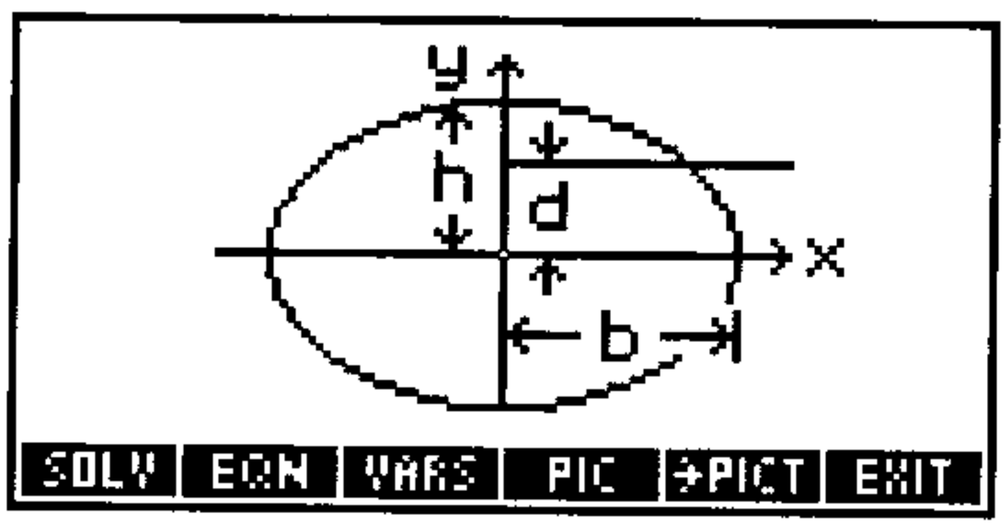
$$J = \frac{\pi \cdot r^4}{2} \qquad Id = I + A \cdot d^2$$

Example:

Given: r=5_cm, d=1.5_cm.

Solution: C=31.4159_cm, A=78.5398_cm^2, I=4908738.5_mm^4, J=9817477.0_mm^4, Id=6675884.4_mm^4.

Ellipse (11, 2)



Equations:

$$A = \pi \cdot b \cdot h \qquad C = 2 \cdot \pi \cdot \sqrt{\frac{b^2 + h^2}{2}} \qquad I = \frac{\pi \cdot b \cdot h^3}{4}$$

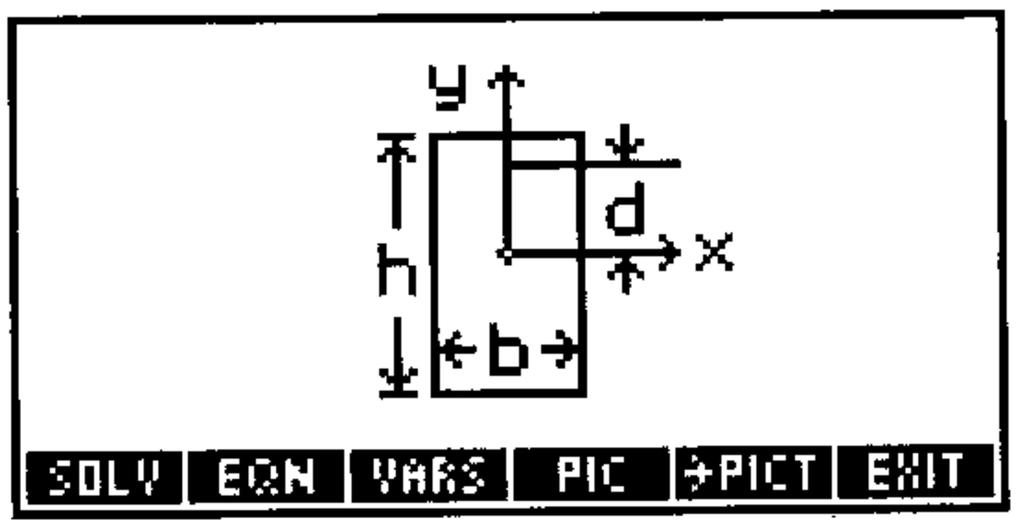
$$J = \frac{\pi \cdot b \cdot h}{4} \cdot \left(b^2 + h^2 \right) \qquad Id = I + A \cdot d^2$$

Example:

Given: b=17.85_μm, h=78.9725_μin, d=.00000012_ft.

Solution: A=1.1249E−6_cm^2, C=7.9805E−3_cm,
I=1.1315E−10_mm^4, J=9.0733E−9_mm^4, Id=1.1330E−10_mm^4.

Rectangle (11, 3)



Equations:

$$A = b \cdot h \qquad P = 2 \cdot b + 2 \cdot h \qquad I = \frac{b \cdot h^3}{12}$$

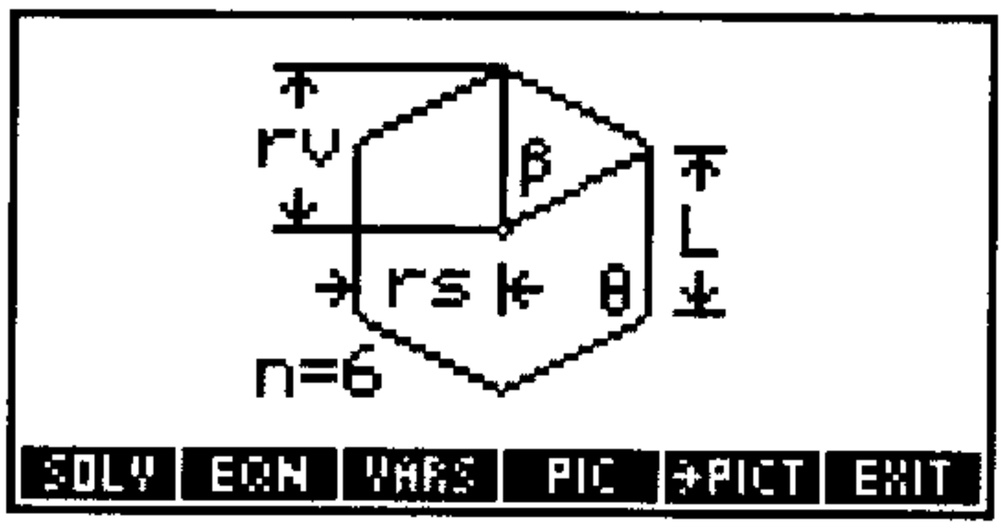
$$J = \frac{b \cdot h}{12} \cdot \left(b^2 + h^2 \right) \qquad Id = I + A \cdot d^2$$

Example:

Given: b=4_chain, h=7_rd, d=39.26_in.
Set guesses for I, J, and Id in km^4.

Solution: A=28328108.2691_cm^2, P=23134.3662_cm,
I=2.9257E−7_km^4, J=1.8211E−6_km^4, Id=2.9539E−7_km^4.

Regular Polygon (11, 4)



Equations:

$$A = \frac{\frac{1}{4} \cdot n \cdot L^2}{TAN\left(\frac{180}{n}\right)} \qquad P = n \cdot L \qquad rs = \frac{\frac{L}{2}}{TAN\left(\frac{180}{n}\right)}$$

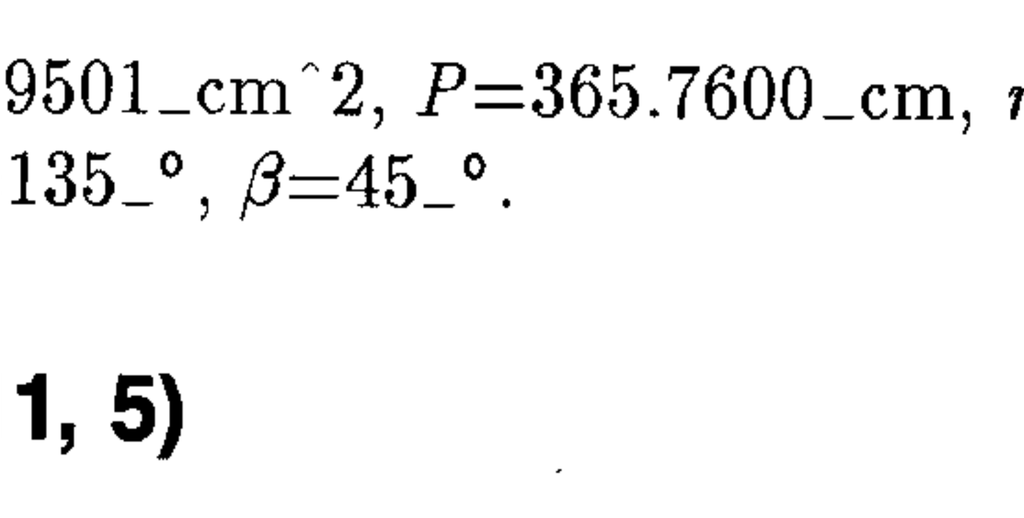
$$rv = \frac{\frac{L}{2}}{SIN\left(\frac{180}{n}\right)} \qquad \theta = \frac{n-2}{n} \cdot 180 \qquad \beta = \frac{360}{n}$$

Example:

Given: n=8, L=.5_yd.

Solution: A=10092.9501_cm^2, P=365.7600_cm, rs=55.1889_cm, rv=59.7361_cm, θ=135_°, β=45_°.

Circular Ring (11, 5)



Equations:

$$A = \pi \cdot \left(ro^2 - ri^2 \right) \qquad I = \frac{\pi}{4} \cdot \left(ro^4 - ri^4 \right)$$

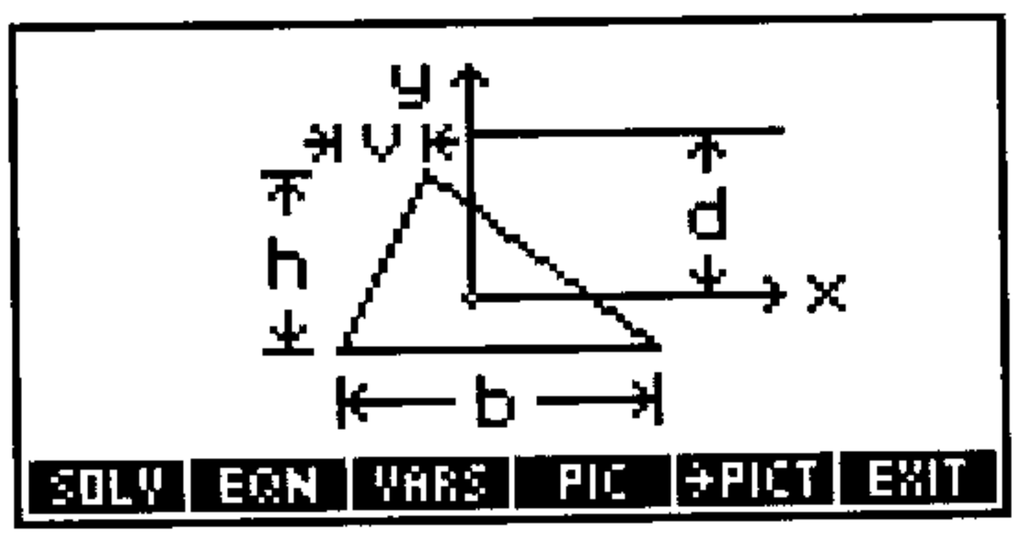
$$J = \frac{\pi}{2} \cdot \left(ro^4 - ri^4 \right) \qquad Id = I + A \cdot d^2$$

Example:

Given: ro=4_μ, ri=25.0_kÅ, d=.1_mil.

Solution: A=3.0631E−7_cm^2, I=1.7038E−10_mm^4, J=3.4076E−10_mm^4, Id=3.0648E−10_mm^4.

Triangle (11, 6)



Equations:

$$A = \frac{b \cdot h}{2} \qquad P = b + \sqrt{v^2 + h^2} + \sqrt{(b - v)^2 + h^2}$$

$$Ix = \frac{b \cdot h^3}{36} \qquad Iy = \frac{b \cdot h}{36} \cdot \left(b^2 - b \cdot v + v^2 \right)$$

$$J = \frac{b \cdot h}{36} \cdot \left(h^2 + b^2 - b \cdot v + v^2 \right) \qquad Id = Ix + A \cdot d^2$$

Example:

Given: h=4.33012781892_in, v=2.5_in, P=15_in, d=2_in.

Solution: b=5.0000_in, Ix=11.2764_in^4, Iy=11.2764_in^4, J=22.5527_in^4, A=10.8253_in^2, Id=54.5776_in^4.

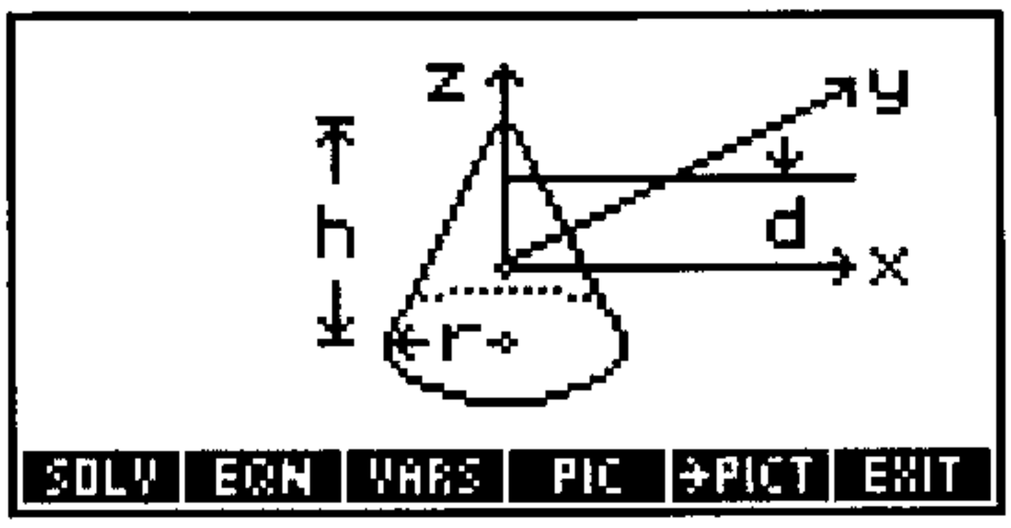
Solid Geometry (12)

Variable Names and Descriptions

A	Total surface area
b	Base length
d	Distance to rotation axis in z direction
h	Height in z direction (Cone, Cylinder), or Height in y direction (Parallelepiped)
I ,Ixx	Moment of inertia about x axis
Id	Moment of inertia in x direction at d
Izz	Moment of inertia about z axis
m	Mass
r	Radius
t	Thickness in z direction
V	Volume

Reference: 4.

Cone (12, 1)



Equations:

$$V = \frac{\pi}{3} \cdot r^2 \cdot h \qquad A = \pi \cdot r^2 + \pi \cdot r \cdot \sqrt{r^2 + h^2}$$

$$Ixx = \frac{3}{20} \cdot m \cdot r^2 + \frac{3}{80} \cdot m \cdot h^2 \qquad Izz = \frac{3}{10} \cdot m \cdot r^2$$

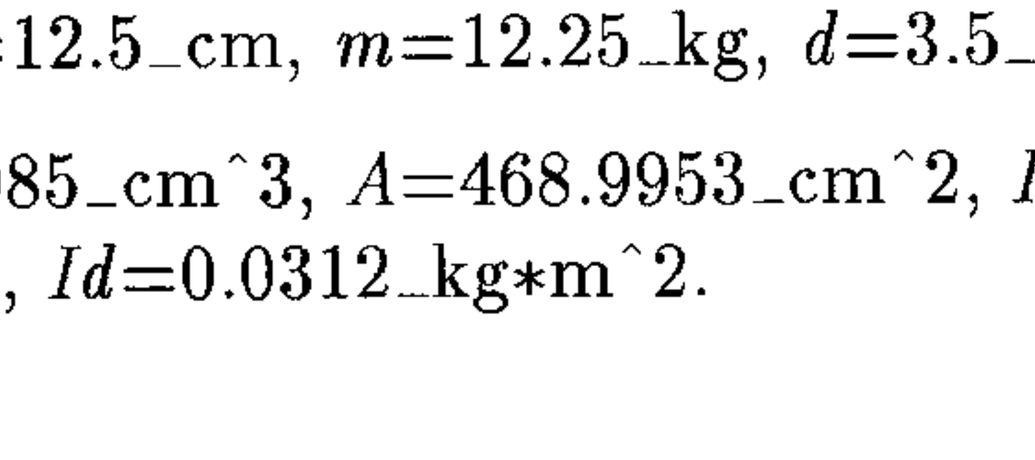
$$Id = Ixx + m \cdot d^2$$

Example:

Given: r=7_cm, h=12.5_cm, m=12.25_kg, d=3.5_cm.

Solution: V=641.4085_cm^3, A=468.9953_cm^2, Ixx=0.0162_kg*m^2, Izz=0.0180_kg*m^2, Id=0.0312_kg*m^2.

Cylinder (12, 2)



Equations:

$$V = \pi \cdot r^2 \cdot h \qquad A = 2 \cdot \pi \cdot r^2 + 2 \cdot \pi \cdot r \cdot h$$

$$Ixx = \frac{1}{4} \cdot m \cdot r^2 + \frac{1}{12} \cdot m \cdot h^2 \qquad Izz = \frac{1}{2} \cdot m \cdot r^2$$

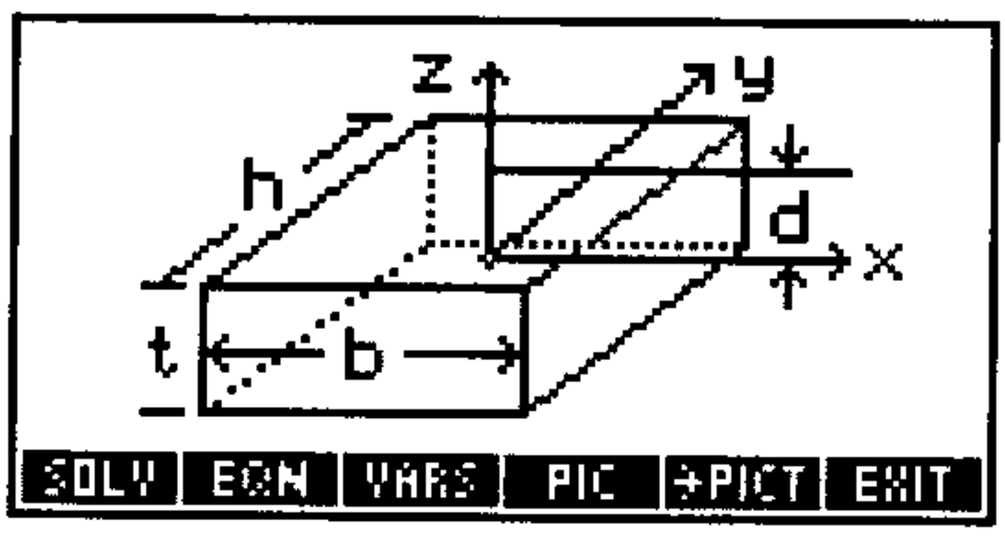
$$Id = Ixx + m \cdot d^2$$

Example:

Given: r=8.5_in, h=65_in, m=12000_lbs, d=2.5_in.

Solution: V=14753.7045_in^3, A=3925.4200_in^2,
Ixx=4441750_lb*in^2, Izz=433500_lb*in^2, Id=4516750_lb*in^2.

Parallelepiped (12, 3)



Equations:

$$V = b \cdot h \cdot t \qquad A = 2 \cdot \left(b \cdot h + b \cdot t + h \cdot t \right)$$

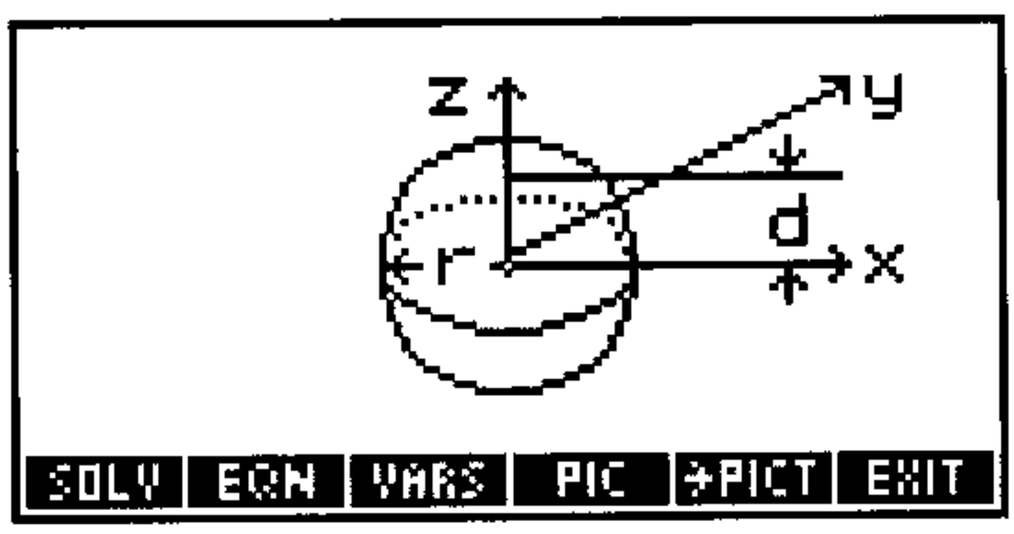
$$I = \frac{1}{12} \cdot m \cdot \left(h^2 + t^2 \right) \qquad Id = I + m \cdot d^2$$

Example:

Given: b=36_in, h=12_in, t=72_in, m=83_lb, d=7_in.

Solution: V=31104_in^3, A=7776_in^2, I=36852_lb*in^2, Id=40919_lb*in^2.

Sphere (12, 4)



Equations:

$$V = \frac{4}{3} \cdot \pi \cdot r^3 \qquad A = 4 \cdot \pi \cdot r^2 \qquad I = \frac{2}{5} \cdot m \cdot r^2 \qquad Id = I + m \cdot d^2$$

Example:

Given: d=14_cm, m=3.75_kg, Id=486.5_lb*in^2.

Solution: r=21.4273_cm, V=41208.7268_cm^3, A=5769.5719_cm^2, I=0.0689_kg*m^2.

Solid State Devices (13)

Variable Names and Descriptions

αF	Forward common-base current gain
αR	Reverse common-base current gain
γ	Body factor
λ	Modulation parameter
μn	Electron mobility
ϕp	Fermi potential
ΔL	Length adjustment (PN Step Junctions), or Channel encroachment (NMOS Transistors)
ΔW	Width adjustment (PN Step Junctions), or Width contraction (NMOS Transistors)
a	Channel thickness
Aj	Effective junction area
BV	Breakdown voltage
Cj	Junction capacitance per unit area
Cox	Silicon dioxide capacitance per unit area
$E1$	Breakdown-voltage field factor
$Emax$	Maximum electric field
$G0$	Channel conductance
gds	Output conductance
gm	Transconductance
I	Diode current
IB	Total base current
IC	Total collector current
$ICEO$	Collector current (collector-to-base open)
ICO	Collector current (emitter-to-base open)
ICS	Collector-to-base saturation current
ID,IDS	Drain current
IE	Total emitter current
IES	Emitter-to-base saturation current

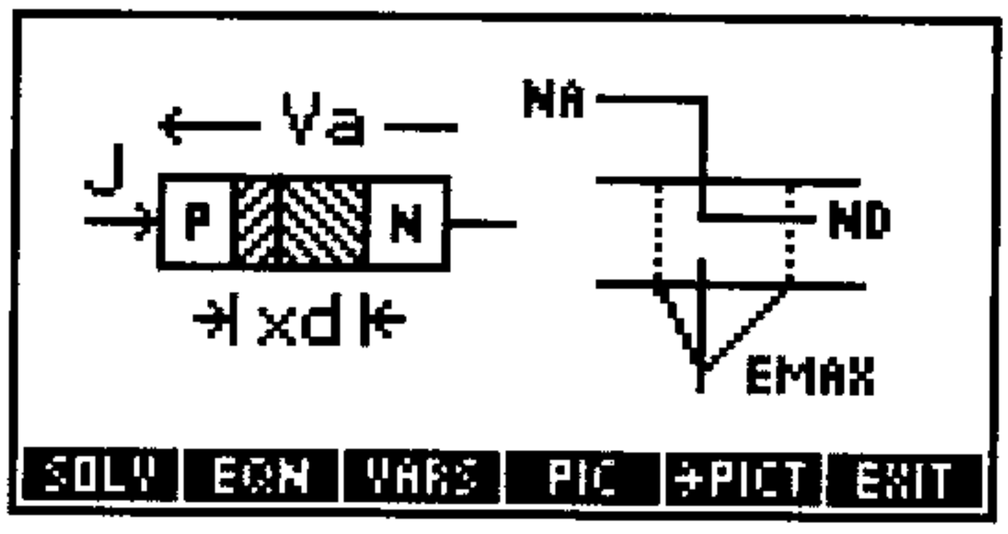
Variable Names and Descriptions (continued)

IS	Transistor saturation current
J	Current density
Js	Saturation current density
L	Drawn mask length (PN Step Junctions), or Drawn gate length (NMOS Transistors), or Channel length (JFETs)
Le	Effective gate length
NA	P-side doping (PN Step Junctions), or Substrate doping (NMOS Transistors)
ND	N-side doping (PN Step Junctions), or N-channel doping (JFETs)
T	Temperature
tox	Gate silicon dioxide thickness
Va	Applied voltage
VBC	Base-to-collector voltage
VBE	Base-to-emitter voltage
Vbi	Built-in voltage
VBS	Substrate voltage
$VCEsat$	Collector-to-emitter saturation voltage
VDS	Applied drain voltage
$VDsat$	Saturation voltage
VGS	Applied gate voltage
Vt	Threshold voltage
$Vt0$	Threshold voltage (at zero substrate voltage)
W	Drawn mask width (PN Step Junctions), or Drawn width (NMOS Transistors), or Channel width (JFETs)
We	Effective width
xd	Depletion-region width
$xdmax$	Depletion-layer width
xj	Junction depth

References: 5, 8.

PN Step Junctions (13, 1)

These equations for a silicon PN-junction diode use a "two-sided step-junction" model—the doping density changes abruptly at the junction. The equations assume the current density is determined by minority carriers injected across the depletion region and the PN junction is rectangular in its layout. The temperature should be between 77 and 500 K. (See "SIDENS" in chapter 3.)



Equations:

$$\mathrm{Vbi} = \frac{\mathrm{k}\cdot\mathrm{T}}{\mathrm{q}}\cdot\mathrm{LN}\left(\frac{\mathrm{NA}\cdot\mathrm{ND}}{\mathrm{ni}^2}\right)$$

$$\mathrm{xd} = \sqrt{\frac{2\cdot\epsilon\mathrm{si}\cdot\epsilon 0}{\mathrm{q}}\cdot(\mathrm{Vbi}-\mathrm{Va})\cdot\left(\frac{1}{\mathrm{NA}}+\frac{1}{\mathrm{ND}}\right)}$$

$$\mathrm{Cj} = \frac{\epsilon\mathrm{si}\cdot\epsilon 0}{\mathrm{xd}} \qquad \mathrm{Emax} = \frac{2\cdot\left(\mathrm{Vbi}-\mathrm{Va}\right)}{\mathrm{xd}}$$

$$\mathrm{BV} = \frac{\epsilon\mathrm{si}\cdot\epsilon 0\cdot\mathrm{E1}^2}{2\cdot\mathrm{q}}\cdot\left(\frac{1}{\mathrm{NA}}+\frac{1}{\mathrm{ND}}\right) \qquad \mathrm{J} = \mathrm{Js}\cdot\left(\mathrm{e}^{\frac{\mathrm{q}\cdot\mathrm{Va}}{\mathrm{k}\cdot\mathrm{T}}}-1\right)$$

$$\mathrm{Aj} = \left(\mathrm{W}+2\cdot\Delta\mathrm{W}\right)\cdot\left(\mathrm{L}+2\cdot\Delta\mathrm{L}\right) + \pi\cdot\left(\mathrm{W}+\mathrm{L}+2\cdot\Delta\mathrm{W}+2\cdot\Delta\mathrm{L}\right)\cdot\mathrm{xj}+2\cdot\pi\cdot\mathrm{xj}^2$$

$$\mathrm{I} = \mathrm{J}\cdot\mathrm{Aj}$$

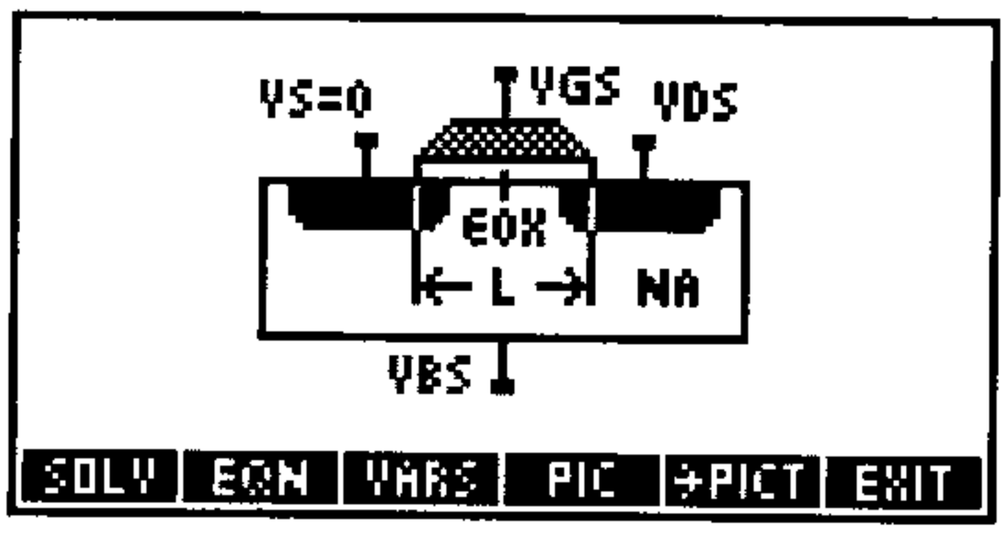
Example:

Given: ND=1E22_cm^−3, NA=1E15_1/cm^3, T=26.85_°C, Js=1E−6_μA/cm^2, Va=−20_V, $E1$=3.3E5_V/cm, W=10_μ, ΔW=1_μ, L=10_μ, ΔL=1_μ, xj=2_μ.

Solution: Vbi=.9962_V, xd=5.2551_μ, Cj=2005.0141_pF/cm^2, $Emax$=79908.5240_V/cm, BV=358.0825_V, J=−1.0E−12_A/cm^2, Aj=3.1993E−6_cm^2, I=−3.1993E−15_mA.

NMOS Transistors (13, 2)

These equations for a silicon NMOS transistor use a two-port network model. They include linear and nonlinear regions in the device characteristics and are based on a gradual-channel approximation (the electric fields in the direction of current flow are small compared to those perpendicular to the flow). The drain current and transconductance calculations differ depending on whether the transistor is in the linear, saturated, or cutoff region. The equations assume the physical geometry of the device is a rectangle, second-order length-parameter effects are negligible, short-channel, hot-carrier, and velocity-saturation effects are negligible, and subthreshold currents are negligible. (See "SIDENS Function" in chapter 3.)



Equations:

$$We = W - 2 \cdot \Delta W \qquad Le = L - 2 \cdot \Delta L \qquad Cox = \frac{\epsilon ox \cdot \epsilon 0}{tox}$$

$$IDS = Cox \cdot \mu n \cdot \left(\frac{We}{Le} \right) \cdot \left((VGS - Vt) \cdot VDS - \frac{VDS^2}{2} \right) \cdot (1 + \lambda \cdot VDS)$$

$$\gamma = \frac{\sqrt{2 \cdot \epsilon si \cdot \epsilon 0 \cdot q \cdot NA}}{Cox}$$

$$Vt = Vt0 + \gamma \cdot \left(\sqrt{2 \cdot ABS(\phi p) + ABS(VBS)} - \sqrt{2 \cdot ABS(\phi p)} \right)$$

$$\phi p = \frac{-k \cdot T}{q} \cdot LN \left(\frac{NA}{ni} \right) \qquad gds = IDS \cdot \lambda$$

$$gm = \sqrt{Cox \cdot \mu n \cdot \left(\frac{We}{Le} \right) \cdot \left(1 + \lambda \cdot VDS \right) \cdot 2 \cdot IDS}$$

$$VDsat = VGS - Vt$$

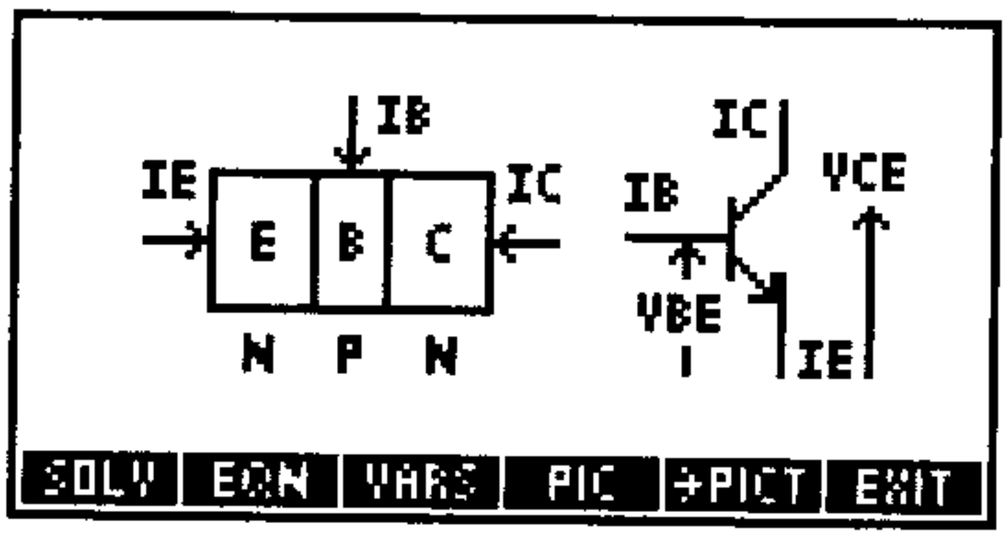
Example:

Given: tox=700_Å, NA=1E15_1/cm^3, μn=600_cm^2/(V*s), T=26.85_°C, $Vt0$=.75_V, VGS=5_V, VBS=0_V, VDS=5_V, W=25_μ, ΔW=1_μ, L=4_m, ΔL=.75_μ, λ=.05_1/V.

Solution: We=23_μ, Le=2.5_μ, Cox=49330.4750_pF/cm^2, γ=.3725_V^.5, ϕp=-.2898_V, Vt=.75_V, $VDsat$=4.25_V, IDS=3.0741_mA, gds=1.5370E-4_S, gm=1.4466_mA/V.

Bipolar Transistors (13, 3)

These equations for an NPN silicon bipolar transistor are based on large-signal models developed by J.J. Ebers and J.L. Moll. The offset-voltage calculation differs depending on whether the transistor is saturated or not. The equations also include the special conditions when the emitter-base or collector-base junction is open, which are convenient for measuring transistor parameters.



Equations:

$$\mathrm{IE} = -\mathrm{IES} \cdot \left(e^{\frac{q \cdot \mathrm{VBE}}{k \cdot T}} - 1 \right) + \alpha\mathrm{R} \cdot \mathrm{ICS} \cdot \left(e^{\frac{q \cdot \mathrm{VBC}}{k \cdot T}} - 1 \right)$$

$$\mathrm{IC} = -\mathrm{ICS} \cdot \left(e^{\frac{q \cdot \mathrm{VBC}}{k \cdot T}} - 1 \right) + \alpha\mathrm{F} \cdot \mathrm{IES} \cdot \left(e^{\frac{q \cdot \mathrm{VBE}}{k \cdot T}} - 1 \right)$$

$$\mathrm{IS} = \alpha\mathrm{F} \cdot \mathrm{IES} \qquad \mathrm{IS} = \alpha\mathrm{R} \cdot \mathrm{ICS} \qquad \mathrm{IB} + \mathrm{IE} + \mathrm{IC} = 0$$

$$\mathrm{ICO} = \mathrm{ICS} \cdot \left(1 - \alpha\mathrm{F} \cdot \alpha\mathrm{R} \right) \qquad \mathrm{ICEO} = \frac{\mathrm{ICO}}{1 - \alpha\mathrm{F}}$$

$$\mathrm{VCEsat} = \frac{k \cdot T}{q} \cdot \mathrm{LN} \left(\frac{1 + \frac{\mathrm{IC}}{\mathrm{IB}} \cdot \left(1 - \alpha\mathrm{R} \right)}{\alpha\mathrm{R} \cdot \left(1 - \frac{\mathrm{IC}}{\mathrm{IB}} \cdot \left(\frac{1 - \alpha\mathrm{F}}{\alpha\mathrm{F}} \right) \right)} \right)$$

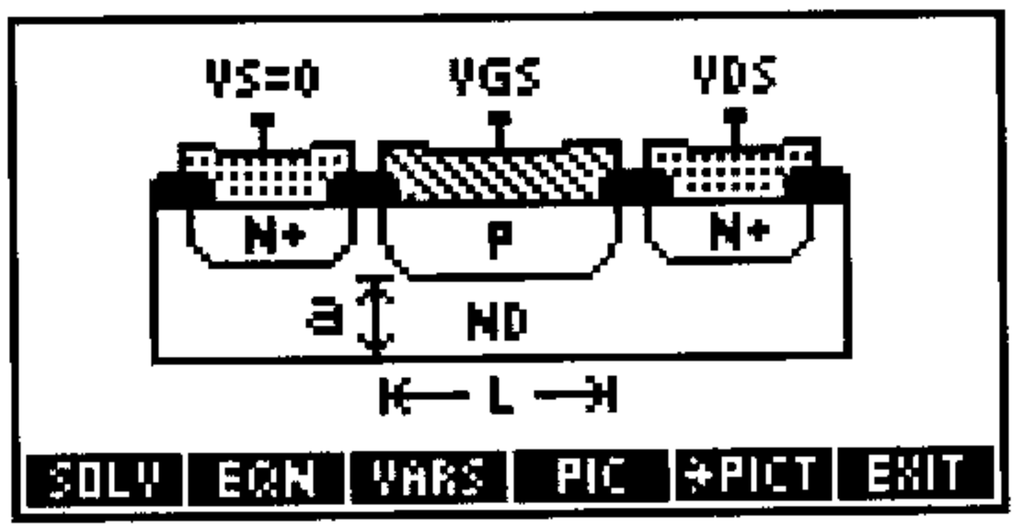
Example:

Given: IES=1E−5_nA, ICS=2E−5_nA, T=26.85_°C, αF=.98, αR=.49, IC=1_mA, VBC=−10_V.

Solution: VBE=.6553_V, IS=0.0000098_nA, ICO=.000010396_nA, $ICEO$=.0005198_nA, IE=−1.0204_mA, IB=.0204_mA, $VCEsat$=0_V.

JFETs (13, 4)

These equations for a silicon N-channel junction field-effect transistor (JFET) are based on the single-sided step-junction approximation, which assumes the gates are heavily doped compared to the channel doping. The drain-current calculation differs depending on whether the gate-junction depletion-layer thickness is less than or greater than the channel thickness. The equations assume the channel is uniformly doped and end effects (such as contact, drain, and source resistances) are negligible. (See "SIDENS" in chapter 3.)



Equations:

$$\mathrm{Vbi} = \frac{\mathrm{k}\cdot\mathrm{T}}{\mathrm{q}} \cdot \mathrm{LN}\left(\frac{\mathrm{ND}}{\mathrm{ni}}\right)$$

$$\mathrm{xdmax} = \sqrt{\frac{2\cdot\epsilon\mathrm{si}\cdot\epsilon 0}{\mathrm{q}\cdot\mathrm{ND}}\cdot\left(\mathrm{Vbi}-\mathrm{VGS}+\mathrm{VDS}\right)}$$

$$\mathrm{G0} = \mathrm{q}\cdot\mathrm{ND}\cdot\mu\mathrm{n}\cdot\left(\frac{\mathrm{a}\cdot\mathrm{W}}{\mathrm{L}}\right)$$

$$\mathrm{ID} = \mathrm{G0}\cdot\left(\mathrm{VDS}-\frac{2}{3}\cdot\sqrt{\frac{2\cdot\epsilon\mathrm{si}\cdot\epsilon 0}{\mathrm{q}\cdot\mathrm{ND}\cdot\mathrm{a}^2}}\cdot\left(\left(\mathrm{Vbi}-\mathrm{VGS}+\mathrm{VDS}\right)^{\frac{3}{2}}-\left(\mathrm{Vbi}-\mathrm{VGS}\right)^{\frac{3}{2}}\right)\right)$$

$$\mathrm{VDsat} = \frac{\mathrm{q}\cdot\mathrm{ND}\cdot\mathrm{a}^2}{2\cdot\epsilon\mathrm{si}\cdot\epsilon 0}-(\mathrm{Vbi}-\mathrm{VGS}) \qquad \mathrm{Vt} = \mathrm{Vbi}-\frac{\mathrm{q}\cdot\mathrm{ND}\cdot\mathrm{a}^2}{2\cdot\epsilon\mathrm{si}\cdot\epsilon 0}$$

$$\mathrm{gm} = \mathrm{G0}\cdot\left(1-\sqrt{\frac{2\cdot\epsilon\mathrm{si}\cdot\epsilon 0}{\mathrm{q}\cdot\mathrm{ND}\cdot\mathrm{a}^2}\cdot(\mathrm{Vbi}-\mathrm{VGS})}\right)$$

Example:

Given: ND=1E16_1/cm^3, W=6_μ, a=1_μ, L=2_μ, μn=1248_cm^2/(V*s), VGS=−4_V, VDS=4_V, T=26.85_°C.

Solution: Vbi=.3493_V, $xdmax$=1.0479_μ, $G0$=5.9986E−4_S, ID=.2268_mA, $VDsat$=3.2537_V, Vt=−7.2537_V, gm=.1462_mA/V.

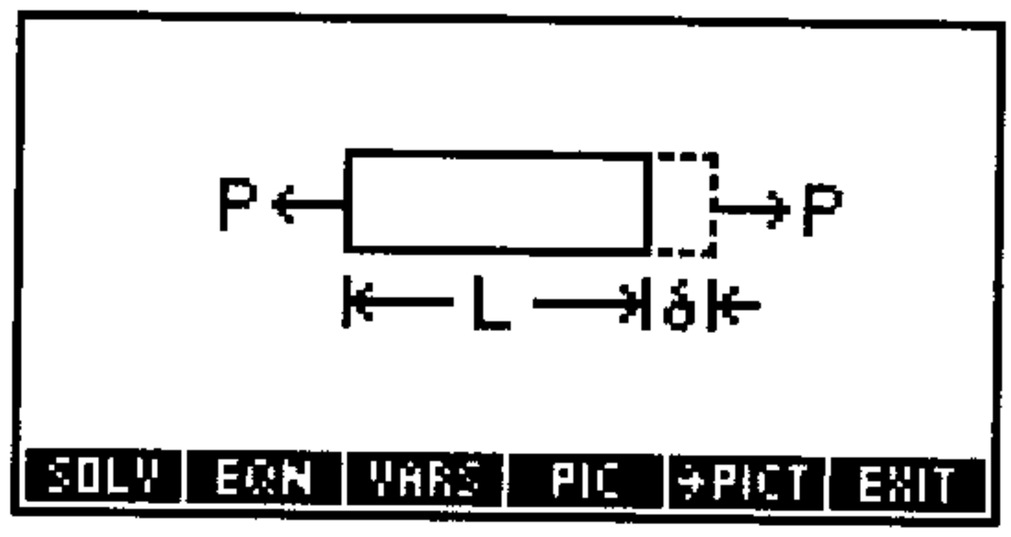
Stress Analysis (14)

Variable Names and Descriptions

δ	Elongation
ϵ	Normal strain
γ	Shear strain
ϕ	Angle of twist
σ	Normal stress
$\sigma 1$	Maximum principal normal stress
$\sigma 2$	Minimum principal normal stress
σavg	Normal stress on plane of maximum shear stress
σx	Normal stress in x direction
$\sigma x1$	Normal stress in rotated-x direction
σy	Normal stress in y direction
$\sigma y1$	Normal stress in rotated-y direction
τ	Shear stress
τmax	Maximum shear stress
$\tau x1y1$	Rotated shear stress
τxy	Shear stress
θ	Rotation angle
$\theta p1$	Angle to plane of maximum principal normal stress
$\theta p2$	Angle to plane of minimum principal normal stress
θs	Angle to plane of maximum shear stress
A	Area
E	Modulus of elasticity
G	Shear modulus of elasticity
J	Polar moment of inertia
L	Length
P	Load
r	Radius
T	Torque

Reference: 2.

Normal Stress (14, 1)



Equations:

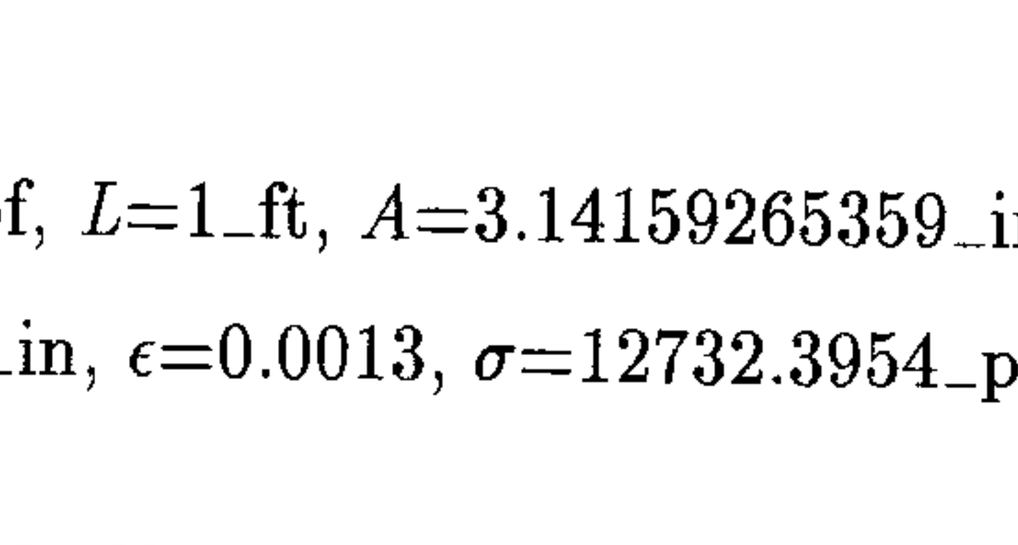
$$\sigma = \mathrm{E} \cdot \epsilon \qquad \epsilon = \frac{\delta}{\mathrm{L}} \qquad \sigma = \frac{\mathrm{P}}{\mathrm{A}}$$

Example:

Given: P=40000_lbf, L=1_ft, A=3.14159265359_in^2, E=10E6_psi.

Solution: δ=0.0153_in, ϵ=0.0013, σ=12732.3954_psi.

Shear Stress (14, 2)



Equations:

$$\tau = \mathrm{G} \cdot \gamma \qquad \gamma = \frac{\mathrm{r} \cdot \phi}{\mathrm{L}} \qquad \tau = \frac{\mathrm{T} \cdot \mathrm{r}}{\mathrm{J}}$$

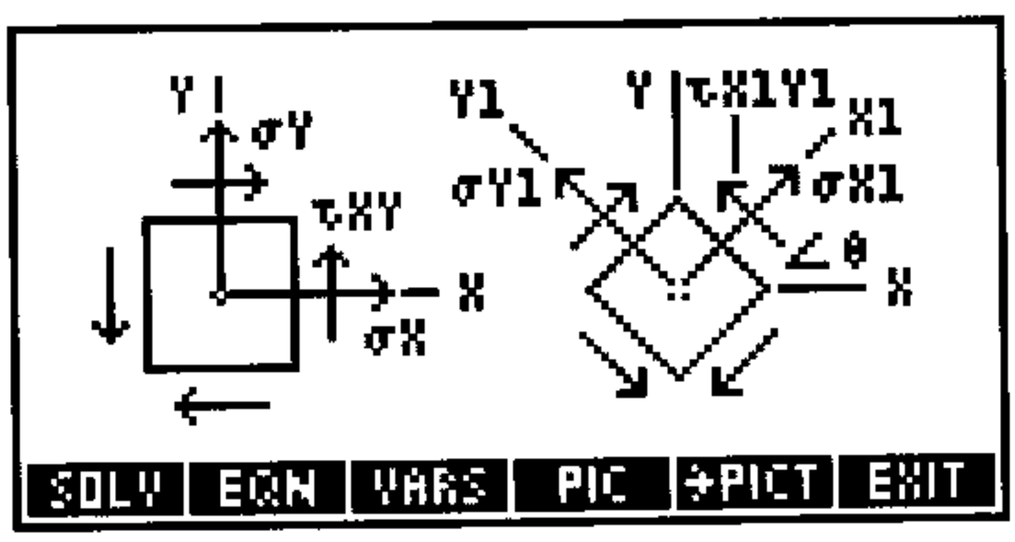
Example:

Given: L=6_ft, r=2_in, J=10.4003897419_in^4, G=12000000_psi, τ=12000_psi.

Solution: T=5200.1949_ft*lbf, ϕ=2.0626_°, γ=5.7296E-2_°.

Stress on an Element (14, 3)

Stresses and strains are positive in the directions shown.



Equations:

$$\sigma x1 = \frac{\sigma x + \sigma y}{2} + \frac{\sigma x - \sigma y}{2} \cdot \mathrm{COS}(2 \cdot \theta) + \tau xy \cdot \mathrm{SIN}(2 \cdot \theta)$$

$$\sigma x1 + \sigma y1 = \sigma x + \sigma y$$

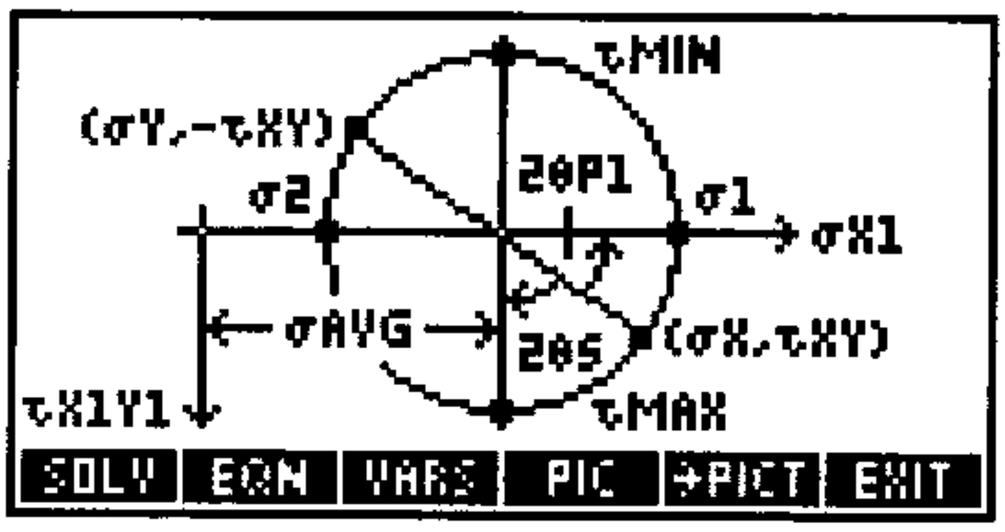
$$\tau x1y1 = -\left(\frac{\sigma x - \sigma y}{2}\right) \cdot \mathrm{SIN}(2 \cdot \theta) + \tau xy \cdot \mathrm{COS}(2 \cdot \theta)$$

Example:

Given: σx=15000_kPa, σy=4755_kPa, τxy=7500_kPa, θ=30_°.

Solution: $\sigma x1$=18933.9405_kPa, $\sigma y1$=821.0595_kPa,
$\tau x1y1$=−686.2151_kPa.

Mohr's Circle (14, 4)



Equations:

$$\sigma 1 = \frac{\sigma x + \sigma y}{2} + \sqrt{\left(\frac{\sigma x - \sigma y}{2}\right)^2 + \tau xy^2}$$

$$\sigma 1 + \sigma 2 = \sigma x + \sigma y$$

$$\mathrm{SIN}(2 \cdot \theta p1) = \frac{\tau xy}{\sqrt{\left(\frac{\sigma x - \sigma y}{2}\right)^2 + \tau xy^2}}$$

$$\theta p2 = \theta p1 + 90 \qquad \tau max = \frac{\sigma 1 - \sigma 2}{2}$$

$$\theta s = \theta p1 - 45 \qquad \sigma avg = \frac{\sigma x + \sigma y}{2}$$

Example:

Given: $\sigma x = -5600_\mathrm{psi}$, $\sigma y = -18400_\mathrm{psi}$, $\tau xy = 4800_\mathrm{psi}$.

Solution: $\sigma 1 = -4000_\mathrm{psi}$, $\sigma 2 = -20000_\mathrm{psi}$, $\theta p1 = 18.4349_^\circ$, $\theta p2 = 108.4349_^\circ$, $\tau max = 8000_\mathrm{psi}$, $\theta s = -26.5651_^\circ$, $\sigma avg = -12000_\mathrm{psi}$.

Waves (15)

Variable Names and Descriptions

β	Sound level
λ	Wavelength
ω	Angular frequency
ρ	Density of medium
B	Bulk modulus of elasticity
f	Frequency
I	Sound intensity
k	Angular wave number
s	Longitudinal displacement at x and t
sm	Longitudinal amplitude
t	Time
v	Speed of sound in medium (Sound Waves), or Wave speed (Transverse Waves, Longitudinal Waves)
x	Position
y	Transverse displacement at x and t
ym	Transverse amplitude

Reference: 3.

Transverse Waves (15, 1)

Equations:

$$\mathrm{y} = \mathrm{ym}\cdot\mathrm{SIN}\left(\mathrm{k}\cdot\mathrm{x} - \omega\cdot\mathrm{t}\right) \qquad \mathrm{v} = \lambda\cdot\mathrm{f} \qquad \mathrm{k} = \frac{2\cdot\pi}{\lambda} \qquad \omega = 2\cdot\pi\cdot\mathrm{f}$$

Example:

Given: ym=6.37_cm, k=32.11_r/cm, x=.03_cm, ω=7000_r/s, t=1_s.

Solution: f=1114.0846_Hz, λ=.0020_cm, y=2.6655_cm, v=218.0006_cm/s.

Longitudinal Waves (15, 2)

Equations:

$$s = sm \cdot COS\left(k \cdot x - \omega \cdot t\right) \qquad v = \lambda \cdot f \qquad k = \frac{2 \cdot \pi}{\lambda} \qquad \omega = 2 \cdot \pi \cdot f$$

Example:

Given: sm=6.37_cm, k=32.11_r/cm, x=.03_cm, ω=7000_r/s, t=1_s.

Solution: s=5.7855_cm, v=2.1800_m/s, λ=.1957_cm, f=1114.0846_Hz.

Sound Waves (15, 3)

Equations:

$$v = \sqrt{\frac{B}{\rho}} \qquad I = \frac{1}{2} \cdot \rho \cdot v \cdot \omega^2 \cdot sm^2$$

$$\beta = 10 \cdot LOG\left(\frac{I}{I0}\right) \qquad \omega = 2 \cdot \pi \cdot f$$

Example:

Given: sm=10_cm, ω=6000_r/s, B=12500_kPa, ρ=65_kg/m^3.

Solution: v=438.5290_m/s, I=5130789412.97_W/m^2, β=217.1018_dB, f=954.9297_Hz.

References

1. Dranchuk, P.M., R.A. Purvis, and D.B. Robinson. "Computer Calculations of Natural Gas Compressibility Factors Using the Standing and Katz Correlation," In *Institute of Petroleum Technical Series*, no. IP 74-008. 1974.

2. Gere, James M., and Stephen P. Timoshenko. *Mechanics of Materials*, 2d ed. PWS Engineering, Boston, 1984.

3. Halliday, David, and Robert Resnick. *Fundamentals of Physics*, 3d ed. John Wiley & Sons, 1988.

4. Meriam, J.L., and L.G. Kraige. *Engineering Mechanics*, 2d ed. John Wiley & Sons, 1986.

5. Muller, Richard S., and Theodore I. Kamins. *Device Electronics for Integrated Cicuits*, 2d ed. John Wiley & Sons, 1986.

6. Serghides, T.K. "Estimate Friction Factor Accurately," In *Chemical Engineering*, Mar. 5, 1984.

7. Siegel, Robert, and John Howell. *Thermal Radiation Heat Transfer*, Vol. 1. National Aeronautics and Space Administration, 1968.

8. Sze, S. *Physics of Semiconductors*, 2d ed. John Wiley & Sons, 1981.

9. Welty, Wicks, and Wilson. *Fundamentals of Momentum, Heat and Mass Transfer*. John Wiley & Sons, 1969.

A

Error and Status Messages

In the following tables, messages are first arranged alphabetically by name and then numerically by message number.

Messages Listed Alphabetically

Message	Meaning	# (hex)
Acknowledged	Alarm acknowledged.	619
All Variables Known	No unknowns to slove for.	E405
Autoscaling	Calculator is autoscaling x- and/or y- axis.	610
Awaiting Server Cmd.	Indicates Server mode active.	C0C
Bad Argument Type	One or more stack arguments were incorrect type for operation.	202
Bad Argument Value	Argument value out of operation's range.	203
Bad Guess(es)	Guess(es) supplied to HP Solve application or ROOT lie outside domain of equation.	A01

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Bad Packet Block check	Computed packet checksum doesn't match checksum in packet.	C01
Can't Edit Null Char.	Attempted to edit a string containing character ■ (character code 0).	102
Circular Reference	Attempted to store a variable name into itself.	129
Connecting	Indicates verifying IR or serial connection.	C0A
Constant?	HP Solve application or ROOT returned same value at every sample point of current equation.	A02
Copied to stack	+⊐TK copied selected equation to stack.	623
Current equation:	Identifies current equation.	608
Deleting Column	MatrixWriter application is deleting a column.	504
Deleting Row	MatrixWriter application is deleting a row.	503
Directory Not Allowed	Name of existing directory variable used as argument.	12A
Directory Recursion	Attempted to store a directory into itself.	002
Empty catalog	No data in current catalog (Equation, Statistics, Alarm).	60D
Empty stack	The stack contains no data.	C15

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Enter alarm, press SET	Alarm entry prompt.	61A
Enter eqn, press NEW	Store new equation in *EQ*.	60A
Enter value (zoom out if >1), press ENTER	Zoom operations prompt.	622
EQ Invalid for MINIT	*EQ* must contain at least two equations (or programs) and two variables.	E403
Extremum	Result returned by HP Solve application or ROOT is an extremum rather than a root.	A06
HALT Not Allowed	A program containing HALT executed while MatrixWriter application, DRAW, or HP Solve application active.	126
I/O setup menu	Identifies I/O setup menu.	61C
Illegal During MROOT	Multiple-Equation Solver. command attempted during MROOT execution.	E406
Implicit () off	Implicit parentheses off.	207
Implicit () on	Implicit parentheses on.	208
Incomplete Subexpression	▶, ▼, or ENTER pressed before all function arguments supplied.	206

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Inconsistent Units	Attempted unit conversion with incompatible units.	B02
Infinite Result	Math exception: Calculation such as 1/0 infinite result.	305
Inserting Column	MatrixWriter application is inserting a column.	506
Inserting Row	MatrixWriter application is inserting a row.	505
Insufficient Memory	Not enough free memory to execute operation.	001
Insufficient Σ Data	A Statistics command was executed when ΣDAT did not contain enough data points for calculation.	603
Interrupted	The HP Solve application or ROOT was interrupted by CANCEL.	A03
Invalid Array Element	ENTER returned object of wrong type for current matrix.	502
Invalid Card Data	HP 48 does not recognize data on plug-in card.	008
Invalid Date	Date argument not real number in correct format, or was out of range.	D01
Invalid Definition	Incorrect structure of equation argument for DEFINE.	12C
Invalid Dimension	Array argument had wrong dimensions.	501

Message	Meaning	# (hex)
Invalid EQ	Attempted operation from PICTURE FCN menu when *EQ* did not contain algebraic, or, attempted DRAW with CONIC plot type when *EQ* did not contain algebraic.	607
Invalid IOPAR	*IOPAR* not a list, or one or more objects in list missing or invalid.	C12
Invalid Mpar	*Mpar* variable not created by MINIT.	E401
Invalid Name	Received illegal filename, or server asked to send illegal filename.	C17
Invalid PPAR	*PPAR* not a list, or one or more objects in list missing or invalid.	12E
Invalid PRTPAR	*PRTPAR* not a list, or one or more objects in list missing or invalid.	C13
Invalid PTYPE	Plot type invalid for current equation.	620
Invalid Repeat	Alarm repeat interval out of range.	D03
Invalid Server Cmd.	Invalid command received while in Server mode.	C08
Invalid Syntax	HP 48 unable execute ENTER, OBJ→, or STR→ due to invalid object syntax.	106

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Invalid Time	Time argument not real number in correct format, or out of range.	D02
Invalid Unit	Unit operation attempted with invalid or undefined user unit.	B01
Invalid User Function	Type or structure of object executed as user-defined function was incorrect.	103
Invalid Σ Data	Statistics command executed with invalid object stored in ΣDAT.	601
Invalid Σ Data LN(Neg)	Non-linear curve fit attempted when ΣDAT matrix contained a negative element.	605
Invalid Σ Data LN(0)	Non-linear curve fit attempted when ΣDAT matrix contained a 0 element.	606
Invalid ΣPAR	ΣPAR not list, or one or more objects in list missing or invalid.	604
Keyword Conflict	A plug-in card conflicts with an equation library variable. Remove the card to continue.	E303
LAST CMD Disabled	(→)(CMD) pressed while that recovery feature disabled.	125
LAST STACK Disabled	(→)(UNDO) pressed while that recovery feature disabled.	124
LASTARG Disabled	(→)(ARG) executed while that recovery feature disabled.	205

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Low Battery	System batteries too low to safely print or perform I/O.	C14
Memory Clear	HP 48 memory was cleared.	005
Name Conflict	Execution of | (where) attempted to assign value to variable of integration or summation index.	13C
Name the equation, press ENTER	Name equation and store it in EQ.	60B
Name the stat data, press ENTER	Name statistics data and store it in ΣDAT.	621
Negative Underflow	Math exception: Calculation returned negative, non-zero result greater than −MINR.	302
No Current Equation	SOLVR, DRAW, or RCEQ executed with nonexistent EQ.	104
No current equation.	Plot and HP Solve application status message.	609
No Picture Available	No picture is included for the selected equation.	E304
No Room in Port	Insufficient free memory in specified RAM port.	00B
No Room to Save Stack	Not enough free memory to save copy of the stack. LAST STACK is automatically disabled.	101

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
No Room to Show Stack	Stack objects displayed by type only due to low memory condition.	131
No stat data to plot	No data stored in ΣDAT.	60F
Non-Empty Directory	Attempted to purge non-empty directory.	12B
Non-Real Result	Execution of HP Solve application, ROOT, DRAW, or $\int$ returned result other than real number or unit.	12F
Nonexistent Alarm	Alarm list did not contain alarm specified by alarm command.	D04
Nonexistent ΣDAT	Statistics command executed when ΣDAT did not exist.	602
Object Discarded	Sender sent an EOF (Z) packet with a "D" in the data field.	C0F
Object In Use	Attempted PURGE or STO into a backup object when its stored object was in use.	009
Object Not in Port	Attempted to access a nonexistent backup object or library.	00C
(OFF SCREEN)	Function value, root, extremum, or intersection was not visible in current display.	61F

A-8 Error and Status Messages

Message	Meaning	# (hex)
Out of Memory	One or more objects must be purged to continue calculator operation.	135
Overflow	Math exception: Calculation returned result greater in absolute value than MAXR.	303
Packet #	Indicates packet number during send or receive.	C10
Parity Error	Received bytes' parity bit doesn't match current parity setting.	C05
Plot Type:	Label introducing current plot type.	61D
Port Closed	Possible I/R or serial hardware failure. Run self-test.	C09
Port Not Available	Used a port command on an empty port, or one containing ROM instead of RAM. Attempted to execute a server command that itself uses the I/O port.	00A
Positive Underflow	Math exception: Calculation returned positive, non-zero result less than MINR.	301
Power Lost	Calculator turned on following a power loss. Memory may have been corrupted.	006

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Processing Command	Indicates processing of host command packet.	C11
Protocol Error	Received a packet whose length was shorter than a null packet. Maximum packet length parameter from other machine is illegal.	C07
Receive Buffer Overrun	Kermit: More than 255 bytes of retries sent before HP 48 received another packet. SRECV: Incoming data overflowed the buffer.	C04
Receive Error	UART overrun or framing error.	C03
Receiving	Identifies object name while receiving.	C0E
Retry #	Indicates number of retries while retrying packet exchange.	C0B
Select a model	Select statistics curve fitting model.	614
Select plot type	Select plot type.	60C
Select repeat interval	Select alarm repeat interval.	61B
Sending	Identifies object name while sending.	C0D

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Sign Reversal	HP Solve application or ROOT unable to find point at which current equation evaluates to zero, but did find two neighboring points at which equation changed sign.	A05
Single Equation	Only one equation supplied to Multiple-Equation Solver. Use HP Solve application.	E402
Timeout	Printing to serial port: Received XOFF and timed out waiting for XON.	C02
	Kermit: Timed out waiting for packet to arrive.	
Too Few Arguments	Command required more arguments than were available on stack.	201
Too Many Unknowns	Multiple Equation Solver can't calculate a value given the current knowns. Supply another value or add an equation.	E404
Transfer Failed	Ten successive attempts to receive a good packet were unsuccessful.	C06
Unable to find root	PROOT is unable to determine all roots of the polynomial.	C001
Unable to Isolate	ISOL failed because specified name absent or contained in argument a function with no inverse.	130

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Undefined Local Name	Executed or recalled local name for which corresponding local variable did not exist.	003
Undefined Name	Executed or recalled global name for which corresponding variable does not exist.	204
Undefined Result	Calculation such as 0/0 generated mathematically undefined result.	304
Undefined XLIB Name	Executed an XLIB name when specified library absent.	004
Warning:	Label introducing current status message.	007
Wrong Argument Count	User-defined function evaluated with an incorrect number of parenthetical arguments.	128
x and y-axis zoom.	Identifies zoom option.	627
x axis zoom.	Identifies zoom option.	625
x axis zoom w/AUTO.	Identifies zoom option.	624
y axis zoom.	Identifies zoom option.	626
ZERO	Result returned by the HP Solve application or ROOT is a root (a point at which current equation evaluates to zero).	A04
" "	Identifies no execution action when EXECUTE pressed.	61E

Messages Listed Numerically

# (hex)	Message
General Messages	
001	Insufficient Memory
002	Directory Recursion
003	Undefined Local Name
004	Undefined XLIB Name
005	Memory Clear
006	Power Lost
008	Invalid Card Data
009	Object In use
00A	Port Not Available
00B	No Room in Port
00C	Object Not in Port
101	No Room to Save Stack
102	Can't Edit Null Char.
103	Invalid User Function
104	No Current Equation
106	Invalid Syntax
124	LAST STACK Disabled
125	LAST CMD Disabled
126	HALT Not Allowed
128	Wrong Argument Count
129	Circular Reference
12A	Directory Not Allowed
12B	Non-Empty Directory
12C	Invalid Definition
12E	Invalid PPAR
12F	Non-Real Result
130	Unable to Isolate
131	No Room to Show Stack
Out-of-Memory Prompts	
135	Out of Memory
13C	Name Conflict

Messages Listed Numerically (continued)

# (hex)	Message
	Stack Errors
201	Too Few Arguments
202	Bad Argument Type
203	Bad Argument Value
204	Undefined Name
205	LASTARG Disabled
	EquationWriter Application Messages
206	Incomplete Subexpression
207	Implicit () off
208	Implicit () on
	Floating-Point Errors
301	Positive Underflow
302	Negative Underflow
303	Overflow
304	Undefined Result
305	Infinite Result
	Array Messages
501	Invalid Dimension
502	Invalid Array Element
503	Deleting Row
504	Deleting Column
505	Inserting Row
506	Inserting Column
	Statistics Messages
601	Invalid Σ Data
602	Nonexistent ΣDAT
603	Insufficient Σ Data
604	Invalid ΣPAR
605	Invalid Σ Data LN(Neg)
606	Invalid Σ Data LN(0)

Messages Listed Numerically (continued)

# (hex)	Message
Plot, I/O, Time and HP Solve Application Messages	
607	Invalid EQ
608	Current equation:
609	No current equation.
60A	Enter eqn, press NEW
60B	Name the equation, press ENTER
60C	Select plot type
60D	Empty catalog
60F	No stat data to plot
610	Autoscaling
614	Select a model
619	Acknowledged
61A	Enter alarm, press SET
61B	Select repeat interval
61C	I/O setup menu
61D	Plot type:
61E	""
61F	(OFF SCREEN)
620	Invalid PTYPE
621	Name the stat data, press ENTER
622	Enter value (zoom out if >1), press ENTER
623	Copied to stack
624	x axis zoom w/AUTO.
625	x axis zoom.
626	y axis zoom.
627	x and y-axis zoom.
A01	Bad Guess(es)
A02	Constant?
A03	Interrupted
A04	Zero
A05	Sign Reversal
A06	Extremum

Messages Listed Numerically (continued)

# (hex)	Message
	Unit Management
B01	Invalid Unit
B02	Inconsistent Units
	I/O and Printing
C01	Bad Packet Block check
C02	Timeout
C03	Receive Error
C04	Receive Buffer Overrun
C05	Parity Error
C06	Transfer Failed
C07	Protocol Error
C08	Invalid Server Cmd.
C09	Port Closed
C0A	Connecting
C0B	Retry #
C0C	Awaiting Server Cmd.
C0D	Sending
C0E	Receiving
C0F	Object Discarded
C10	Packet #
C11	Processing Command
C12	Invalid IOPAR
C13	Invalid PRTPAR
C14	Low Battery
C15	Empty Stack
C17	Invalid Name

Messages Listed Numerically (continued)

# (hex)	Message
Time Messages	
D01	Invalid Date
D02	Invalid Time
D03	Invalid Repeat
D04	Nonexistent Alarm
Equation Library Messages	
E303	Keyword Conflict
E304	No Picture Available
Multiple-Equation Solver Messages	
E401	Invalid Mpar
E402	Single Equation
E403	EQ Invalid for MINIT
E404	Too Many Unknowns
E405	All Variables Known
E406	Illegal During MROOT
Miscellaneous Messages	
70000	(user-defined message created with DOERR)

B

Table of Units

HP 48 Units

Unit (Full Name)	Value in SI Units
a (are)	100 m^2
A (ampere)	1 A
acre (acre)	4046.87260987 m^2
arcmin (minute of arc)	2.90888208666 x 10^{-4} r
arcs (second of arc)	4.8481368111 x 10^{-6} r
atm (atmosphere)	101325 $kg/m \cdot s^2$
au (astronomical unit)	1.495979 x 10^{11} m
Å (Angstrom)	1 x 10^{-10} m
b (barn)	1 x 10^{-28} m^2
bar (bar)	100000 $kg/m \cdot s^2$
bbl (barrel)	.158987294928 m^3
Bq (becquerel)	1 1/s
Btu (international table Btu)	1055.05585262 $kg \cdot m^2/s^2$
bu (bushel)	.03523907 m^3
°C (degree Celsius)	1 K or 274.15 K
c (speed of light)	299792458 m/s
C (coulomb)	1 A·s
cal (calorie)	4.1868 $kg \cdot m^2/s^2$
cd (candela)	1 cd
chain (chain)	20.1168402337 m
Ci (curie)	3.7 x 10^{10} 1/s
ct (carat)	.0002 kg
cu (US cup)	2.365882365 x 10^{-4} m^3
° (degree)	1.74532925199 x 10^{-2} r
d (day)	86400 s

HP 48 Units (continued)

Unit (Full Name)	Value in SI Units
dyn (dyne)	.00001 $kg \cdot m/s^2$
erg (erg)	.0000001 $kg \cdot m^2/s^2$
eV (electron volt)	$1.60217733 \times 10^{-19}$ $kg \cdot m^2/s^2$
F (farad)	1 $A^2 \cdot s^4/kg \cdot m^2$
°F (degrees Fahrenheit)	0.555555555556 K or 255.927777778 K
fath (fathom)	1.82880365761 m
fbm (board foot)	.002359737216 m^3
fc (footcandle)	10.7639104167 $cd \cdot sr/m^2$
Fdy (faraday)	96487 $A \cdot s$
fermi (fermi)	1×10^{-15} m
flam (footlambert)	3.42625909964 cd/m^2
ft (international foot)	.3048 m
ftUS (survey foot)	.304800609601 m
g (gram)	.001 kg
ga (standard freefall)	9.80665 m/s^2
gal (US gallon)	.003785411784 m^3
galC (Canadian gallon)	.0045460$9m^3$
galUK (UK gallon)	.004546092 m^3
gf (gram-force)	.00980665 $kg \cdot m/s^2$
grad (gradient)	$1.57079632679 \times 10^{-2}$ r
grain (grain)	.0006479891 kg
Gy (gray)	1 m^2/s^2
H (henry)	1 $kg \cdot m^2/A^2 \cdot s^2$
h (Hour)	3600 s
hp (horsepower)	745.699871582 $kg \cdot m^2/s^3$
Hz (hertz)	$1/s$
in (inch)	.0254 m
inHg (inches of mercury, 0°C)	3386.38815789 $kg/m \cdot s^2$
inH20 (inches of water, 60°F)	248.84 $kg/m \cdot s^2$
J (joule)	1 $kg \cdot m^2/s^2$
K (kelvins)	1 K
kg (kilogram)	1 kg

Unit (Full Name)	Value in SI Units
kip (kilopound-force)	4448.22161526 kg·m/s^2
knot (nautical miles per hour)	.514444444444 m/s
kph (kilometers per hour)	.277777777778 m/s
l (liter)	.001 m^3
lam (lambert)	3183.09886184 cd/m^2
lb (avoirdupois pound)	.45359237 kg
lbf (pound-force)	4.44822161526 kg·m/s^2
lbt (troy pound)	.3732417216 kg
lm (lumen)	1 cd·sr
lx (lux)	1 cd·sr/m^2
lyr (light year)	9.46052840488 × 10^{15} m
m (meter)	1 m
μ (micron)	1 × 10^{-6} m
mho (mho)	1 A^2·s^3/kg·m^2
mi (international mile)	1609.344 m
mil (mil)	.0000254 m
min (minute)	60 s
miUS (US statute mile)	1609.34721869 m
mmHg (millimeter of mercury (torr), 0°C)	133.322368421 kg/m·s^2
mol (mole)	1 mol
mph (miles per hour)	.44704 m/s
N (newton)	1 kg·m/s^2
nmi (nautical mile)	1852 m
Ω (ohm)	1 kg·m^2/A^2·s^3
oz (ounce)	.028349523125 kg
ozfl (US fluid ounce)	2.95735295625 × 10^{-5} m^3
ozt (troy ounce)	.0311034768 kg
ozUK (UK fluid ounce)	2.8413075 × 10^{-5} m^3
P (poise)	.1 kg/m·s
Pa (pascal)	1 kg/m·s^2
pc (parsec)	3.08567818585 × 10^{16} m
pdl (poundal)	.138254954376 kg·m/s^2
ph (phot)	10000 cd·sr/m^2
pk (peck)	.0088097675 m^3

HP 48 Units (continued)

Unit (Full Name)	Value in SI Units
`psi` (pounds per square inch)	6894.75729317 $kg/m \cdot s^2$
`pt` (pint)	.000473176473 m^3
`qt` (quart)	.000946352946 m^3
`r` (radian)	1 r
`R` (roentgen)	.000258 $A \cdot s/kg$
`°R` (degrees Rankine)	0.555555555556 K
`rad` (rad)	.01 m^2/s^2
`rd` (rod)	5.02921005842 m
`rem` (rem)	.01 m^2/s^2
`s` (second)	1 s
`S` (siemens)	1 $A^2 \cdot s^3/kg \cdot m^2$
`sb` (stilb)	10000 cd/m^2
`slug` (slug)	14.5939029372 kg
`sr` (steradian)	1 sr
`st` (stere)	1 m^3
`St` (stokes)	.0001 m^2/s
`Sv` (sievert)	1 m^2/s^2
`t` (metric ton)	1000 kg
`T` (tesla)	1 $kg/A \cdot s^2$
`tbsp` (tablespoon)	$1.47867647813 \times 10^{-5}$ m^3
`therm` (EEC therm)	105506000 $kg \cdot m^2/s^2$
`ton` (short ton)	907.18474 kg
`tonUK` (long ton (UK))	1016.0469088 kg
`torr` (torr (mmHg))	133.322368421 $kg/m \cdot s^2$
`tsp` (teaspoon)	$4.92892159375 \times 10^{-6}$ m^3
`u` (unified atomic mass)	$1.6605402 \times 10^{-27}$ kg
`V` (volt)	1 $kg \cdot m^2/A \cdot s^3$
`W` (watt)	1 $kg \cdot m^2/s^3$
`Wb` (weber)	1 $kg \cdot m^2/A \cdot s^2$
`yd` (international yard)	.9144 m
`yr` (year)	31556925.9747 s

C

System Flags

This appendix lists the HP 48 system flags. You can set, clear, and test all flags. The default state of the flags is *clear*—except for the Binary Integer Wordsize flags (flags −5 through −10).

System Flags

Flag	Description
−1	Principal Solution. *Clear*: QUAD and ISOL return a result representing all possible solutions. *Set*: QUAD and ISOL return only the principal solution.
−2	Symbolic Constants. *Clear*: Symbolic constants (e, i, π, MAXR, and MINR) retain their symbolic form when evaluated, unless the Numerical Results flag −3 is set. *Set*: Symbolic constants evaluate to numbers, regardless of the state of the Numerical Results flag −3.
−3	Numerical Results. *Clear*: Functions with symbolic arguments, including symbolic constants, evaluate to symbolic results. *Set*: Functions with symbolic arguments, including symbolic constants, evaluate to numbers.
−4	Not used.
−5 thru −10	Binary Integer Wordsize. Combined states of flags −5 through −10 set the wordsize from 1 to 64 bits.

System Flags (continued)

Flag	Description
−11 and −12	Binary Integer Base. HEX: −11 *set*, −12 *set*. DEC: −11 *clear*, −12 *clear*. OCT: −11 *set*, −12 *clear*. BIN: −11 *clear*, −12 *set*.
−13	Not used.
−14	Financial Payment Mode. *Clear:* TVM calculations assume end-of-period payments. *Set:* TVM calculations assume beginning-of-period payments.
−15 and −16	Rectangular: −16 *clear*. Polar/Cylindrical: −15 *clear*, −16 *set*. Polar/Spherical: −15 *set*, −16 *set*.
−17 and −18	Degrees: −17 *clear*, −18 *clear*. Radians: −17 *set*. Grads: −17 *clear*, −18 *set*.
−19	*Clear*:→V2 and creates a 2-dimensional vector from 2 real numbers. *Set*:→V2 and creates a complex number from 2 real numbers.
−20	Underflow Exception. *Clear*: Underflow exception returns 0, sets flag −23 or −24. *Set*: Underflow exception treated as an error.
−21	Overflow Exception. *Clear*: Overflow exception returns ±9.99999999999E499 and sets flag −25. *Set*: Overflow exception treated as an error.
−22	Infinite Result Exception. *Clear*: Infinite result exception treated as an error. *Set*: Infinite result exception returns ±9.99999999999E499 and sets flag −26.
−23 −24 −25	Negative Underflow Indicator. Positive Underflow Indicator. Overflow Indicator.
−26	Infinite Result Indicator. When an exception occurs, corresponding flag (−23 through −26) is set only if the exception is *not* treated as an error.

Flag	Description
−27	Display of symbolic complex numbers. *Clear*: Displays symbolic complex numbers in coordinate form (i.e. `'(x,y)'`). *Set*: Displays symbolic complex numbers using `'i'` (i.e. `'x+y*i'`).
−28	Simultaneous Plotting of Multiple Functions. *Clear*: Multiple equations are plotted serially. *Set*: Multiple equations are plotted simultaneously.
−29	Draw Axes. *Clear*: Axes are drawn for two-dimensional and statistical plots. *Set*: Axes are not drawn for two-dimensional and statistical plots.
−30	Not used.
−31	Curve Filling. *Clear*: Curve filling between plotted points enabled. *Set*: Curve filling between plotted points suppressed.
−32	Graphics Cursor. *Clear*: Graphics cursor always dark. *Set*: Graphics cursor dark on light background and light on dark background.
−33	I/O Device. *Clear*: I/O directed to serial port. *Set*: I/O directed to IR port.
−34	Printing Device. *Clear*: Printer output directed to IR printer. *Set*: Printer output directed to serial port if flag −33 is clear.
−35	I/O Data Format. *Clear*: Objects transmitted in ASCII form. *Set*: Objects transmitted in binary (memory image) form.

System Flags (continued)

Flag	Description
−36	I/O Receive Overwrite. *Clear*: If file name received by HP 48 matches existing HP 48 variable name, new variable name with number extension is created to prevent overwrite. *Set*: If file name received by HP 48 matches existing HP 48 variable name, existing variable is overwritten.
−37	Double-Spaced Printing. *Clear*: Single-spaced printing. *Set*: Double-spaced printing.
−38	Line Feed. *Clear*: Linefeed added at end of each print line. *Set*: No linefeed added at end of each print line.
−39	I/O Messages. *Clear*: I/O messages displayed. *Set*: I/O messages suppressed.
−40	Clock Display. *Clear*: Clock displayed only when TIME menu selected. *Set*: Ticking clock displayed at all times.
−41	Clock Format. *Clear*: 12-hour clock. *Set*: 24-hour clock.
−42	Date Format. *Clear*: MM/DD/YY (month/day/year) format. *Set*: DD.MM.YY (day.month.year) format.
−43	Repeat Alarms Not Rescheduled. *Clear*: Unacknowledged repeat appointment alarms automatically rescheduled. *Set*: Unacknowledged repeat appointment alarms not rescheduled.
−44	Acknowledged Alarms Saved. *Clear*: Acknowledged appointment alarms deleted from alarm list. *Set*: Acknowledged appointment alarms saved in alarm list.

System Flags (continued)

Flag	Description
−45 thru −48	Number of Decimal Digits. Combined states of flags −45 through −48 sets number of decimal digits in Fix, Scientific, and Engineering modes.
−49 and −50	Number Display Format. Standard: −49 *clear*, −50 *clear*. Fix: −49 *set*, −50 *clear*. Scientific: −49 *clear*, −50 *set*. Engineering: −49 *set*, −50 *set*.
−51	Fraction Mark. *Clear*: Fraction mark is . (period). *Set*: Fraction mark is , (comma).
−52	Single-Line Display. *Clear*: Display gives preference to object in level 1, using up to four lines of stack display. *Set*: Display of object in level 1 restricted to one line.
−53	Precedence. *Clear*: Certain parentheses in algebraic expressions suppressed to improve legibility. *Set*: All parentheses in algebraic expressions displayed.
−54	Tiny Array Elements. *Clear*: Singular values computed by RANK (and other commands that compute the rank of a matrix) that are more than 1×10^{-14} times smaller than the largest computed singular value in the matrix are converted to zero. Automatic rounding for DET is enabled. *Set*: Small computed singular values (see above) not converted. Automatic rounding for DET is disabled.
−55	Last Arguments. *Clear*: Command arguments saved. *Set*: Command arguments not saved.
−56	Error Beep. *Clear*: Error and BEEP-command beeps enabled. *Set*: Error and BEEP-command beeps suppressed.

Flag	Description
−57	Alarm Beep. *Clear*: Alarm beep enabled. *Set*: Alarm beep suppressed.
−58	Verbose Messages. *Clear*: Parameter variable data automatically displayed. *Set*: Automatic display of parameter variable data is suppressed.
−59	Fast Browser Display. *Clear*: Variable Browser shows variable names and contents. *Set*: Variable Browser shows variable names only.
−60	Alpha Lock. *Clear*: Single-Alpha activated by pressing [α] once. Alpha lock activated by pressing [α] twice. *Set*: Alpha lock activated by pressing [α] once. (Single-Alpha not available.)
−61	User-Mode Lock. *Clear*: 1-User mode activated by pressing [↰][USER] once. User mode activated by pressing [↰][USER] twice. *Set*: User mode activated by pressing [↰][USER] once. (1-User mode not available.)
−62	User Mode. *Clear*: User mode not active. *Set*: User mode active.
−63	Vectored [ENTER]. *Clear*: [ENTER] evaluates command line. *Set*: User-defined [ENTER] activated.
−64	Index Wrap Indicator. *Clear*: Last execution of GETI or PUTI did not increment index to first element. *Set*: Last execution of GETI or PUTI did increment index to first element.

D

Reserved Variables

The HP 48 uses the following *reserved variables*. These have specific purposes, and their names are used as implicit arguments for certain commands. Avoid using these variables' names for other purposes, or you may interfere with the execution of the commands that use these variables.

You can change some of the values in these variables with programmable commands, while others require you to store new values into the appropriate place.

Reserved Variable	What It Contains	Used By
ALRMDAT	Alarm parameters.	TIME ALRM operations
CST	List defining the CST (custom) menu.	MENU, CST
"*der*"-names	User-defined derivative.	∂
EQ	Current equation.	ROOT, DRAW
EXPR	Current expression.	SYMBOLIC
IOPAR	I/O parameters.	I/O commands
MHpar	Minehunt game status.	MINEHUNT
Mpar	Multiple-Equation Solver equations.	EQ LIB
n1, n2, ...	Arbitrary integers.	ISOL, QUAD
Nmines	Minehunt game data.	MINEHUNT

Reserved Variable	What It Contains	Used By
PPAR	Plotting parameters.	DRAW
PRTPAR	Printing parameters.	PRINT commands
s1, s2, ...	Arbitrary signs.	ISOL, QUAD
VPAR	Viewing parameters.	DRAW
ZPAR	Plot zoom factors.	DRAW
ΣDAT	Statistical data.	Statistics application, DRAW
ΣPAR	Statistical parameters.	Statistics application, DRAW

Contents of the Reserved Variables

Most reserved variables (except *ALRMDAT*, *IOPAR* and *PRTPAR*) can be stored with different contents in different directories. This allows you, for example, to save several sets of statistical data in different directories.

ALRMDAT

ALRMDAT does not reside in a particular directory. You cannot access the variable itself, but you can access its data from any directory using the RCLALARM and STOALARM commands, or through the Alarm Catalog.

ALRMDAT contains a list of these alarm parameters:

Parameter (Command)	Description	Default Value
date (→DATE)	A real number specifying the date of the alarm: *MM.DDYYYY* (or *DD.MMYYYY* if flag −42 is set). If *YYYY* is not included, the current year is used.	Current date.
time (→TIME)	A real number specifying the time of the alarm: *HH.MMSS*.	00.0000
action	A string or object: ■ a string creates an *appointment alarm*, which beeps and displays the string ■ any other object creates a *control alarm*, which executes the object	Empty string (appointment alarm).
repeat	A real number specifying the interval between automatic recurrences of the alarm, given in ticks (a tick is $^{1}/_{8192}$ of a second).	0

Parameters without commands can be modified with a program by storing new values in the list contained in *ALRMDAT* (use the PUT command).

CST

CST contains a list (or a name specifying a list) of the objects that define the CST (*custom*) menu. Objects in the custom menu behave as do objects in built-in menus. For example:

- Names behave like the VAR menu keys. Thus, if *ABC* is a variable name, HEC evaluates *ABC*, → HEC recalls its contents, and ← HEC stores new contents in *ABC*.
- The menu label for the name of a directory has a bar over the left side of the label; pressing the menu key switches to that directory.
- Unit objects act like unit catalog entries (and have left-shifted conversion capabilities, for example).

- String keys echo the string.
- You can include backup objects in the list defining a custom menu by tagging the name of the backup object with its port location (0 through 33).

You can specify menu labels and key actions independently by replacing a single object within the custom-menu list with a list of the form { "*label-object*" *action-object* }. (See "Customizing Menus" and "Enhancing Custom Menus" in chapter 30 of the *HP 48 User's Guide* for more information.)

To provide different shifted actions for custom menu keys, *action-object* can be a list containing three action objects in this order:

- The unshifted action (required if you want to specify the shifted actions).
- The left-shifted action.
- The right-shifted action.

See "Enhancing Custom Menus" in chapter 30 of the *HP 48 User's Guide*.

"der-" Names

If ∂ is applied to a function for which there is no built-in derivative, it returns a new function whose name is "der" followed by the original function name. These "der"-function names are reserved variable names.

For an example, refer to "Creating User-Defined Derivatives" in chapter 20 of the *HP 48 User's Guide*.

EQ

EQ contains the current equation or the name of the variable containing the current equation.

EQ supplies the equation for ROOT, as well as for the plotting command DRAW when the plot type is FUNCTION, CONIC, POLAR, PARAMETER, TRUTH, or DIFFEQ. (ΣDAT supplies the information when the plot type is HISTOGRAM, BAR, or SCATTER.)

The object in *EQ* can be an algebraic object, a number, a name, or a program. How DRAW interprets *EQ* depends on the plot type.

For graphics use, *EQ* can also be a list of equations or other objects. If *EQ* contains a list, then DRAW treats each object in turn as the current equation, and plots them successively. However, ROOT in the HP Solve application *cannot* solve an *EQ* containing a list.

To alter the contents of *EQ*, use the command STEQ.

EXPR

EXPR contains the current algebraic expression (or the name of the variable containing the current expression) used by the SYMBOLIC application and its associated commands. The object in *EQ* must be an algebraic or a name.

IOPAR

IOPAR is a variable in the *HOME* directory that contains a list of the I/O parameters needed for a communications link with a computer. It is created the first time you transfer data or open the serial port (OPENIO), and is automatically updated whenever you change the I/O settings. All *IOPAR* parameters are integers.

Parameter (Command)	Description	Default Value
baud (BAUD)	The baud rate: 1200, 2400, 4800, or 9600.	9600
parity (PARITY)	The parity used: 0=none, 1=odd, 2=even, 3=mark, 4=space. The value can be positive or negative: a positive parity is used upon both transmit and receive; a negative parity is used only upon transmit.	0

Parameter (Command)	Description	Default Value
receive pacing	Controls whether receive pacing is used: a nonzero real value enables pacing, while zero disables it. Receive pacing sends an XOFF signal when the receive buffer is almost full, and sends an XON signal when it can take more data again. Pacing is not used for Kermit I/O, but is used for other serial I/O transfers.	0 (no pacing)
transmit pacing	Controls whether transmit pacing is used: a nonzero real value enables pacing, while zero disables it. Transmit pacing stops transmission upon receipt of XOFF, and resumes transmission upon receipt of XON. Pacing is not used for Kermit I/O, but is used for other serial I/O transfers.	0 (no pacing)
checksum (CKSM)	Error-detection scheme requested when initiating SEND: ■ 1=1-digit arithmetic checksum ■ 2=2-digit arithmetic checksum ■ 3=3-digit cyclic redundancy check.	3
translation code (TRANSIO)	Controls which characters are translated: ■ 0=none ■ 1=translate character 10 (line feed only) to/from characters 10 and 13 (line feed and carriage return) ■ 2=translate characters with numbers 128 through 159 (80–9F hex) ■ 3=translate characters with numbers 128 through 255.	1

Parameters without commands can be modified with a program by storing new values in the list contained in *IOPAR* (use the PUT command), or by editing *IOPAR* directly.

D-6 Reserved Variables

MHpar

MHpar stores the status of an interrupted Minehunt game. *MHpar* is created when you exit Minehunt by pressing (STO). If *MHpar* still exists when you restart Minehunt, the interrupted game resumes and *MHpar* is purged.

Mpar

Mpar is created when you use the Equation Library's Multiple-Equation Solver, and it stores the set of equations you're using.

When the Equation Library starts the Multiple-Equation Solver, it first stores a list of the equation set in *EQ*, and stores the equation set, a list of variables, and additional information in *Mpar*. *Mpar* is then used to set up the Solver menu for the current equation set.

Mpar is structured as library data dedicated to the Multiple Equation Solver application. This means that although you can view and edit *Mpar* directly, you can edit it only indirectly by executing commands that modify it.

You can also use the MINIT command ((←) EQ LIB MES MINIT) to create *MHpar* from a set of equations on the stack. See "Defining a Set of Equations" in chapter 25 of the *HP 48 User's Guide.*

n1, n2, ...

The ISOL and QUAD commands return *general* solutions (as opposed to *principal* solutions) for operations. A general solution contains variables for arbitrary integers or arbitrary signs, or both.

The variable *n1* represents an arbitrary integer 0, ±1, ±2, etc. Additional arbitrary integers are represented by *n2*, *n3*, etc.

If flag −1 is set, then ISOL and QUAD return principal solutions, in which case the arbitrary integer is always zero.

Nmines

Nmines is a variable you create in the current directory to control the number of mines used in the Minehunt game. *Nmines* contains an integer in the range 1 to 64; if *Nmines* is negative, the mines are visible during the game.

PPAR

PPAR is a variable in the current directory. It contains a list of plotting parameters used by the DRAW command for all mathematical and statistical plots, by AUTO for autoscaling, and by the interactive (nonprogrammable) graphics operations.

Parameter (Command)	Description	Default Value
$(x_{\min}, y_{\min})$ (XRNG, YRNG)	A complex number specifying the lower left corner of *PICT* (the lower left corner of the display range).	$(-6.50, -3.1)$
$(x_{\max}, y_{\max})$ (XRNG, YRNG)	A complex number specifying the upper right corner of *PICT* (the upper right corner of the display range).	(6.5, 3.2)
indep (INDEP)	A name specifying the independent variable, or a list containing that name and two numbers that specify the minimum and maximum values for the independent variable (the plotting range).	*X*

Parameter (Command)	Description	Default Value
res (RES)	Resolution. A real number specifying the interval between values of the independent variable. For plots of equations, this determines the plotting interval along the x-axis. A binary number specifies the *pixel* resolution (how many columns of pixels between points). An integer specifies the resolution in *user* units (how many user units between points). Resolution for statistical plots is different (see below).	0
axes (AXES)	A complex number specifying the user-unit coordinates of the plot origin, or a list containing the following: ■ the complex number specifying the origin ■ a real number, binary integer, or list containing two real numbers or binary integers specifying the tick-mark annotation (see ATICK) ■ two strings specifying labels for the horizontal and vertical axes	(0, 0)
ptype (BAR, etc.)	A command name specifying the plot type (BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, or YSLICE).	FUNCTION

Parameter (Command)	Description	Default Value
depend (DEPND)	A name specifying the dependent variable, or a list containing the name and two numbers that specify vertical plotting range. For DIFFEQ, the second element of the list may also be a real vector that represents the initial value.	*Y*

Parameters without commands can be modified with a program by storing new values in the list contained in *PPAR* (use the PUT command).

The RESET operation (◀ PLOT PPAR RESET) resets the *PPAR* parameters (except *ptype*) to their default values, and erases *PICT*.

Note that *res* behaves differently for the statistical plot types BAR, HISTOGRAM, and SCATTER than for other plot types. For BAR, *res* specifies bar width; for HISTOGRAM, *res* specifies bin width; *res* does not affect SCATTER.

PRTPAR

PRTPAR is a variable in the *HOME* directory that contains a list of printing parameters. It is created automatically the first time you use a printing command.

Parameter (Command)	Description	Default Value
delay time (DELAY)	A real number, in the range 0 to 6.9, specifying the number of seconds the HP 48 waits between sending lines. This should be at least as long as the time required to print the longest line. If the delay is too short for the printer, you will lose data. The delay setting also affects serial printing if transmit-pacing (in *IOPAR*) is not being used.	1.8
remap (OLDPRT stores the character-remapping string for the HP 82240A Infrared Printer)	A string defining the current remapping of the extended character set for printing. The string can contain as many characters as you want to remap, with the first character being the new character 128, the second being the new charcter 129, etc. (Any character number that exceeds the string length will not be remapped.) See the example below.	Empty string.
line length	A real number specifying the number of characters in a line for serial printing. This does *not* affect infrared printing.	80
line termination	A string specifying the line-termination method for serial printing. This does *not* affect infrared printing.	Control characters 13 (carriage return) and 10 (line feed).

Parameters without commands can be modified with a program by storing new values in the list contained in *PRTPAR* (use the PUT command).

A change in a parameter is effective immediately, *except* when printing the display using the simultaneous keystrokes [ON] [I/O] (because this does not use *PRTPAR*). This printing method is affected only by the delay parameter, a change in which will not affect [ON] [I/O] until after the next printing command has been executed. To use a new delay time with [ON] [I/O] immediately, use the DELAY command.

Example: If the remapping string were "ABCDEFGH" and the character to be printed had value 131, then the character actually printed would be "D", since 131−128=3 and "A" has the value zero. A character code of 136 or greater would not be remapped since 136−128=8, which exceeds the length of the string.

s1, s2, ...

The ISOL and QUAD commands return *general* solutions (as opposed to *principal* solutions) for operations. A general solution contains variables for arbitrary integers or arbitrary signs or both.

The variable *s1* represents an arbitrary + or − sign. Additional arbitrary signs are represented by *s2*, *s3*, etc.

If flag −1 is set, then ISOL and QUAD return principal solutions, in which case the arbitrary sign is always +1.

VPAR

VPAR is a variable in the current directory. It contains a list of parameters used by the 3D plot types. The main data structure stored in VPAR describes the "view volume," the abstract three-dimensional region in which the function is plotted.

Parameter (Command)	Description	Default Value
$(x_{\text{left}}, x_{\text{right}})$ (XVOL)	Real numbers that specify the width of the view volume.	(−1, 1)
$(y_{\text{far}}, y_{\text{near}})$ (YVOL)	Real numbers that specify the depth of the view volume.	(−1, 1)
$(z_{\text{low}}, z_{\text{high}})$ (ZVOL)	Real numbers that specify the height of the view volume.	(−1, 1)
$(x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}})$ (EYEPT)	Real numbers that specify the point in space from which the plot is viewed.	(0, −3, 0)
$(x_{\text{step}}, y_{\text{step}})$ (NUMX,NUMY)	Real numbers that specify the increments between of x-coordinates and y-coordinates plotted. The increments are equal to the range for the axes divided by the number of steps. Used instead of (or in combination with) *res*.	(10, 8)
$(xx_{\text{left}}, xx_{\text{right}})$ (XXRNG)	Real numbers that specify the width of the input plane (domain). Used by GRIDMAP and PARSURFACE.	(−1, 1)
$(yy_{\text{far}}, yy_{\text{near}})$ (YYRNG)	Real numbers that specify the depth of the input plane (domain). Used by GRIDMAP and PARSURFACE.	(−1, 1)

Parameters without commands can be modified programmatically by storing new values in the list contained in *VPAR* (use the PUT command).

The RESET operation (◀ PLOT NXT 3D VPAR NXT RESET) resets the *VPAR* parameters to their default values.

ZPAR

ZPAR is a variable in the current directory. It contains a list of zooming parameters used by the DRAW command for all 2-D mathematical and statistical plots.

Parameter (Command)	Description	Default Value
h-factor	Real number that specifies the horizontal zoom factor.	4
v-factor	Real number that specifies the vertical zoom factor.	4
recenter flag	0 or 1 depending on whether the recenter at crosshairs option was selected in the set zoom factors input form.	0
{ *list* }	An empty list, or a copy of the last *PPAR*.	

Use the set zoom factors input form (ZFACT) to modify *ZPAR*.

ΣDAT

ΣDAT is a variable in the current directory that contains either the current statistical matrix or the name of the variable containing this matrix. This matrix contains the data used by the Statistics applications.

Statistical Matrix for Variables 1 to m

var_1	var_2	...	var_m
x_{11}	x_{21}	...	x_{m1}
x_{12}	x_{22}	...	x_{m2}
⋮	⋮	⋮	⋮
x_{1n}	x_{2n}	...	x_{mn}

You can designate a new current statistical matrix by entering new data, editing the current data, or selecting another matrix.

The command CLΣ clears the current statistical matrix.

ΣPAR

ΣPAR is a variable in the current directory that contains either the current statistical parameter list or the name of the variable containing this list.

Parameter (Command)	Description	Default Value
$column_{\text{indep}}$ (XCOL)	A real number specifying the independent-variable's column number.	1
$column_{\text{dep}}$ (YCOL)	A real number specifying the dependent-variable's column number.	2
intercept (LR)	A real number specifying the coefficient of intercept as determined by the current regression.	0
slope (LR)	A real number specifying the coefficient of slope as determined by the current regression.	0
model (LINFIT, etc.)	A command specifying the regression model (LINFIT, EXPFIT, PWRFIT, or LOGFIT).	LINFIT

E

New Commands

In the following tables, new commands (commands that were not available on a standard HP 48S series calculator) are arranged alphabetically and followed by brief descriptions. All of these commands are described in chapter 3.

New Commands Listed Alphabetically

Command	Brief Description
ADD	Adds list elements.
AMORT	Amortizes a loan or investment based upon the current amortization settings.
ANIMATE	Displays graphic objects in sequence.
ATICK	Sets the axes tick-mark annotation in the reserved variable *PPAR*.
CHOOSE	Creates a user-defined choose box.
CLTEACH	Removes the EXAMPLES subdirectory and its contents from the HOME directory.
COL+	Inserts an array (vector or matrix) into a matrix.
COL−	Deletes a column from of a matrix.
COL→	Transforms a series of column vectors and a column count into a matrix, or transforms a sequence of numbers and an element count into a vector.
→COL	Transforms a matrix into a series of column vectors, or transforms a vector into its elements.

Command	Brief Description
COND	Returns the 1-norm (column norm) condition number of a square matrix.
CONLIB	Opens the Constants Library catalog.
CONST	Returns the value of a constant.
CSWP	Swaps columns in a matrix.
CYLIN	Sets Cylindrical coordinate mode.
DARCY	Calculates the Darcy friction factor of certain fluid flows.
DIAG→	Takes an array and a specified dimension and returns a matrix whose main diagonal elements are the elements of the array.
→DIAG	Returns a vector that contains the major diagonal elements of a matrix.
DIFFEQ	Specifies differential equations as the plot type.
DOLIST	Applies commands, programs, or user-defined functions to lists.
DOSUBS	Applies a program or command to groups of elements in a list.
EGV	Computes the eigenvalues and right eigenvectors for a square matrix.
EGVL	Computes the eigenvalues of a square matrix.
ENDSUB	Provides a way to access the total number of sublists used while executing a program or command using DOSUBS.
EQNLIB	Starts the Equation Library application.
EYEPT	Specifies the coordinates of the eye point in a perspective plot.

New Commands Listed Alphabetically (continued)

Command	Brief Description
F0λ	Returns the fraction of total black-body emissive power.
FANNING	Calculates the Fanning friction factor of certain fluid flows.
FFT	Computes the one- or two-dimensional discrete Fourier transform of an array.
FREE1	Frees the previously merged RAM in port 1.
GRIDMAP	Specifies grid mapping as the plot type.
HEAD	Returns the first element of a list or string.
IFFT	Computes the one- or two-dimensional inverse discrete Fourier transform of a vector or matrix.
INFORM	Creates a user-defined dialog box (Input Form).
LIBEVAL	Evaluates unnamed library objects by their memory addresses.
LININ	Tests whether an algebraic is structurally linear for a given variable.
ΣLIST	Returns the sum of the elements in a list.
ΠLIST	Returns the product of the elements in a list.
ΔLIST	Returns the set of first differences.
LQ	Returns the LQ factorization of an $n \times m$ matrix.
LSQ	Returns the minimum norm least squares solution to any system of linear equations.
LU	Returns the LU decomposition of a square matrix.
MCALC	Designates a variable as a calculated value (not user-defined).
MERGE1	Merges the RAM from the card in port 1 with the rest of main user memory.

Command	Brief Description
MINEHUNT	Starts the MINEHUNT game.
MINIT	Creates the reserved variable *Mpar*.
MITM	Changes multiple equation menu titles and order.
MROOT	Solves for one or more variables.
MSGBOX	Creates a user-defined message box.
MSOLVR	Gets the Multiple-Equation Solver variable menu for the set of equations defined by *Mpar*.
MUSER	Designates a variable as user-defined.
NDIST	Returns the normal probability distribution.
NOVAL	Place holder for reset and initial values in user-defined dialog boxes.
NSUB	Provides a way to access the current sublist number during an iteration of a program or command applied using DOSUBS.
NUMX	Sets the number of x-steps for each y-step in 3D perspective plots.
NUMY	Sets the number of y-steps across the view volume in 3D perspective plots.
PARSURFACE	Specifies 3D parameterized surface grip mapping as the plot type.
PCOEF	Returns the coefficients of a monic polynomial.
PCOV	Calculates population covariance.
PCONTOUR	Specifies pseudo-contour as the plot type.
PEVAL	Evaluates an n-degree polynomial at x.
PINIT	Initializes the plug-in card ports.
PROOT	Returns all roots of an n-degree polynomial having real or complex coefficients.

Command	Brief Description
PSDEV	Calculates population standard deviation.
PVAR	Calculates population variance.
QR	Returns the QR factorization of an $n \times m$ matrix.
RANK	Returns the rank of a rectangular matrix.
RANM	Returns a matrix of random integers.
RCI	Multiplies a row of a matrix by a constant.
RCIJ	Multiplies a row of a matrix by a constant, and then adds the product to another row of the matrix.
RECT	Sets Rectangular coordinate mode.
REVLIST	Reverses the order of the elements in a list.
RKF	Computes the solution to an initial value problem for a differential equation, using the Runge-Kutta-Fehlberg method.
RKFERR	Returns the absolute error estimate for a given step when solving an initial value problem for a differential equation (using RKF method).
RKFSTEP	Computes the next solution step to an initial value problem for a differential equation.
ROW+	Inserts an array into a matrix.
ROW−	Deletes a row from a matrix.
RREF	Converts a rectangular matrix to reduced row echelon form.
RRK	Computes the solution to an initial value problem for a differential equation with known partial derivatives.
RRKSTEP	Computes the next solution step to an initial value problem for a differential equation, and displays the method used to arrive at that result.

New Commands Listed Alphabetically (continued)

Command	Brief Description
RSBERR	Returns an error estimate for a given step when solving an initial values problem for a differential equation (using the Rosenbrock method).
RSWP	Swaps rows in a matrix.
SCHUR	Returns the Schur decomposition of a square matrix.
SEQ	Returns a list of results generated by repeatedly executing an object on a specified range of elements.
SIDENS	Calculates the intrinsic density of silicon as a function of temperature.
SLOPEFIELD	Specifies slopefield as the plot type.
SNRM	Returns the spectral norm of an array.
SOLVEQN	Starts the solver for a specified set of equations.
SORT	Sorts the elements in a list in ascending order.
SPHERE	Sets Spherical coordinate mode.
SRAD	Returns the spectral radius of a square matrix.
STREAM	Applies an object to every element in a list.
SVD	Returns the singular value decomposition of an $n \times m$ matrix.
SVL	Returns the singular values of an $m \times n$ matrix.
TAIL	Returns all but the first element of a list or string.
TDELTA	Calculates a temperature change.
TEACH	Creates an EXAMPLES subdirectory in the HOME directory and loads HP 48 programming, graphing, and solver examples from ROM into it.
TINC	Calculates a temperature increment.
TRACE	Returns the trace of a square matrix.

New Commands Listed Alphabetically (continued)

Command	Brief Description
TVM	Start the TVM solver.
TVMBEG	Specifies that payments are made at the beginning of compounding periods.
TVMEND	Specifies that payments are made at the end of compounding periods.
TVMROOT	Solves for the specified TVM variable using values from the remaining TVM variables.
VERSION	Returns the software version and copyright message.
WIREFRAME	Specifies wireframe as the plot type.
XRECV	Receives an object via XModem.
XSEND	Sends an object via XModem.
XVOL	Sets the width of the view volume in the reserved variable *VPAR*.
XXRNG	Specifies the x range of an input plane (domain) for GRIDMAP and PARSURFACE plots.
YSLICE	Specifies y-slice cross sections as the plot type.
YVOL	Sets the depth of the view volume in the reserved variable *VPAR*.
YYRNG	Specifies the y range of an input plane (domain) for GRIDMAP and PARSURFACE plots.
ZFACTOR	Calculates the the gas compressibility correction factor for nonideal behavior of a hydrocarbon gas.
ZVOL	Sets the height of the view volume in the reserved variable *VPAR*.

F

Technical Reference

This appendix contains the following information:

- Object sizes.
- Mathematical simplification rules used by the HP 48.
- Symbolic differentiation patterns used by the HP 48.
- The EquationWriter's expansion rules.
- References used as sources for constants and equations in the HP 48 (other than those in the Equation Library).

Object Sizes

The following table lists object types and their size in bytes. (Note that characters in names, strings, and tags use 1 byte each.)

Object Size

Object Type	Size (bytes)
Algebraic	5 + size of included objects
Backup Object	12 + number of name characters + size of included object
Binary Integer	13
Command	2.5
Complex matrix	15 + 16 × number of elements
Complex number	18.5
Complex vector	12.5 + 16 × number of elements
Directory	6.5 + size of included variables
Graphics Object	10 + number of rows × CEIL(columns/8)
List	5 + size of included objects
Matrix	15 + 8 × number of elements
Program	12.5 + size of included objects
Quoted global or local name	8.5 + number of characters
Real number	10.5
String	5 + number of characters
Tagged Object	3.5 + number of tag characters + size of untagged object
Unit Object	7.5 +
real magnitude	2.5 or 10.5
each prefix	6
each unit name	5 + number of characters
each ×, ^, or /	2.5
each exponent	2.5 or 10.5
Unquoted global or local name	3.5 + number of characters
Vector	12.5 + 8 × number of elements
XLIB name	5.5

Automatic Simplification Rules

The following tables list the automatic simplification rules for the HP 48.

Addition and Subtraction

Object	Simplified
$x-x$	0
$0+x$	x
$(0,0)+x$	x
$0-x$	NEG(x)
$(0,0)-x$	NEG(x)
$x+0$	x

Object	Simplified
$x+(0,0)$	x
$x+-\mathrm{p}$	$x-\mathrm{p}$
$x-0$	x
$x-(0,0)$	x
$x--\mathrm{p}$	$x+\mathrm{p}$

Multiplication and Division

Object	Simplified
INV(i)	$-i$
$y\times$INV(x)	y/x
$y/$INV(x)	$y\times x$
$0\times x$	0
$(0,0)\times x$	$(0,0)$
$i\times i$	-1
$1\times x$	x
$(1,0)\times x$	x
$(-1)\times x$	NEG(x)
$(-1,0)\times x$	NEG(x)
$x\times 0$	0
$x\times(0,0)$	$(0,0)$
$x\times 1$	x

Object	Simplified
$x\times(1,0)$	x
$x\times(-1)$	NEG(x)
$x\times(-1,0)$	NEG(x)
$x/1$	x
$x/(1,0)$	x
$x/(-1)$	NEG(x)
$x/(-1,0)$	NEG(x)
$0/x$	0
$(0,0)/x$	$(0,0)$
$1/x$	INV(x)
$(1,0)x$	INV(x)
$(-1)/x$	$-$INV(x)
$(-1,0)/x$	$-$INV(x)

Powers

Object	Simplified
1^x	1
$(1,0)^x$	$(1,0)$
$\mathrm{SQ}(\sqrt{}(x))$	x
$\mathrm{SQ}(y^x)$	$y^{(2\times x)}$
$\mathrm{SQ}(i)$	-1
x^0	1
$x^{(0,0)}$	$(1,0)$
x^1	x

Object	Simplified
$x^{(1,0)}$	x
$x^{(-1)}$	$\mathrm{INV}(x)$
$x^{(-1,0)}$	$\mathrm{INV}(x)$
$(\sqrt{}x)^2$	x
$(\sqrt{}x)^{(2,0)}$	x
i^2	-1
$i^{(2,0)}$	$(-1,0)$

Parts

Object	Simplified
$\mathrm{ABS}(\mathrm{ABS}(x))$	$\mathrm{ABS}(x)$
$\mathrm{ABS}(\mathrm{NEG}(x))$	$\mathrm{ABS}(x)$
$\mathrm{CONJ}(\mathrm{CONJ}(x))$	x
$\mathrm{CONJ}(\mathrm{IM}(x))$	$\mathrm{IM}(x)$
$\mathrm{CONJ}(\mathrm{RE}(x))$	$\mathrm{RE}(x)$
$\mathrm{CONJ}(i)$	$-i$
$\mathrm{IM}(\mathrm{CONJ}(x))$	$-\mathrm{IM}(x)$
$\mathrm{IM}(\mathrm{IM}(x))$	0
$\mathrm{IM}(\mathrm{RE}(x))$	0
$\mathrm{IM}(\mathrm{p})$	0
$\mathrm{IM}(i)$	1
$\mathrm{MAX}(x,x)$	x

Object	Simplified
$\mathrm{MIN}(x,x)$	x
$\mathrm{MOD}(0,x)$	0
$\mathrm{MOD}(x,x)$	0
$\mathrm{MOD}(x,0)$	x
$x\ \mathrm{MOD}\ y\ \mathrm{MOD}\ y$	$x\ \mathrm{MOD}\ y$
$\mathrm{RE}(\mathrm{CONJ}(x))$	$\mathrm{RE}(x)$
$\mathrm{RE}(\mathrm{IM}(x))$	$\mathrm{IM}(x)$
$\mathrm{RE}(\mathrm{RE}(x))$	$\mathrm{RE}(x)$
$\mathrm{RE}(\pi)$	π
$\mathrm{RE}(i)$	0
$\mathrm{SIGN}(\mathrm{SIGN}(x))$	$\mathrm{SIGN}(x)$

Symbolic Integration Patterns

This table lists the symbolic integration patterns used by the HP 48. These are the integrands that the HP 48 can integrate symbolically.

ϕ is a linear function of the variable of integration. The antiderivatives should be divided by the first-order coefficient in ϕ to reduce the expression to its simplest form. Also, patterns beginning with 1/ match INV: for example, $1/\phi$ is the same as INV(ϕ).

Symbolic Integration

Pattern	Antiderivative
ACOS(ϕ)	$\phi\times$ACOS(ϕ)$-\sqrt{(1-\phi^2)}$
ALOG(ϕ)	.434294481904$\times$ALOG(ϕ)
ASIN(ϕ)	$\phi\times$ASIN(ϕ)$+\sqrt{(1-\phi^2)}$
ATAN(ϕ)	$\phi\times$ATAN($\phi-$LN$(1+\phi^2)/2$
COS(ϕ)	SIN(ϕ)
1/(COS(ϕ)$\times$SIN(ϕ))	LN(TAN(ϕ))
COSH(ϕ)	SINH(ϕ)
1/(COSH(ϕ)$\times$SINH(ϕ))	LN(TANH(ϕ))
1/(COSH(ϕ)^2)	TANH(ϕ)
EXP(ϕ)	EXP(ϕ)
EXPM(ϕ)	EXP(ϕ)$-\phi$
LN(ϕ)	$\phi\times$LN(ϕ)$-\phi$
LOG(ϕ)	.434294481904$\times\phi\times$LN(ϕ)$-\phi$
SIGN(ϕ)	ABS(ϕ)
SIN(ϕ)	$-$COS(ϕ)
1/(SIN(ϕ)$\times$COS(ϕ))	LN(TAN(ϕ))
1/(SIN(ϕ)$\times$TAN(ϕ))	$-$INV(SIN(ϕ))
1/(SIN(ϕ)$\times$TAN(ϕ))	$-$INV(SIN(ϕ))
1/(SIN(ϕ)^2)	$-$INV(TAN(ϕ))
SINH(ϕ)	COSH(ϕ)
1/(SINH(ϕ)$\times^2$	$-$INV(SIN(ϕ))

Symbolic Integration (continued)

Pattern	Antiderivative
$1/(\mathrm{SINH}(\phi)\times\mathrm{COSH}(\phi))$	$\mathrm{LN}(\mathrm{TANH}(\phi))$
$1/(\mathrm{SINH}(\phi)\times\mathrm{TANH}(\phi))$	$-\mathrm{INV}(\mathrm{SINH}(\phi))$
$\mathrm{SQ}(\phi)$	$\phi^3/3$
$\mathrm{TAN}(\phi)^2$	$\mathrm{TAN}(\phi)-\phi$
$\mathrm{TAN}(\phi)$	$-\mathrm{LN}(\mathrm{COS}(\phi))$
$\mathrm{TAN}(\phi)/\mathrm{COS}(\phi)$	$\mathrm{INV}(\mathrm{COS}(\phi))$
$1/\mathrm{TAN}(\phi)$	$\mathrm{LN}(\mathrm{SIN}(\phi))$
$1/\mathrm{TAN}(\phi)\times\mathrm{SIN}(\phi))$	$-\mathrm{INV}(\mathrm{SIN}(\phi))$
$\mathrm{TANH}(\phi)$	$\mathrm{LN}(\mathrm{COSH}(\phi))$
$\mathrm{TANH}(\phi)/\mathrm{COSH}(\phi)$	$\mathrm{INV}(\mathrm{COSH}(\phi))$
$1/\mathrm{TANH}(\phi)$	$\mathrm{LN}(\mathrm{SINH}(\phi))$
$1/\mathrm{TANH}(\phi)\times\mathrm{SINH}(\phi))$	$-\mathrm{INV}(\mathrm{SINH}(\phi))$
$\sqrt{\phi}$	$2\times\phi^{1.5}/3$
$1/\sqrt{\phi}$	$2\times\sqrt{\phi}$
$1/(2\times\sqrt{(\phi)})$	$2\times\sqrt{(\phi)}\times.5$
ϕ^z (z symbolic)	$\mathrm{IFTE}(z==-1,\mathrm{LN}(\phi),\phi^{(z+1)}/(z+1))$
ϕ^z (z real, $\neq 0,-1$)	$\phi^{(z+1)}/(z+1)$
ϕ^0	ϕ
ϕ^{-1}	$\mathrm{LN}(\phi)$
$1/\phi$	$\mathrm{LN}(\phi)$
$1/(1-\phi^2)$	$\mathrm{ATANH}(\phi)$
$1/(1+\phi^2)$	$\mathrm{ATAN}(\phi)$
$1/(\phi^2+1)$	$\mathrm{ATAN}(\phi)$
$1/(\sqrt{(\phi-1)}\times\sqrt{(\phi+1)})$	$\mathrm{ACOSH}(\phi)$
$1/\sqrt{(1-\phi^2)}$	$\mathrm{ASIN}(\phi)$
$1/\sqrt{(1+\phi^2)}$	$\mathrm{ASINH}(\phi)$
$1/\sqrt{(\phi^2+1)}$	$\mathrm{ASINH}(\phi)$

Trigonometric Expansions

The following tables list expansions for trigonometric functions in Radians mode when using the →DEF, TRG*, and →TRG operations. These operations appear in the EquationWriter RULES menu.

→DEF Expansions

Function	Expansion
SIN(x)	$\frac{EXP(x \times i) - EXP(-(x \times i))}{2 \times i}$
COS(x)	$\frac{EXP(x \times i) + EXP(-(x \times i))}{2}$
TAN(x)	$\frac{EXP(x \times i \times 2) - 1}{(EXP(x \times i \times 2) + 1) \times i}$
SINH(x)	$-(SIN(x \times i) \times i)$
COSH(x)	$COS(x \times i)$
TANH(x)	$TAN(x \times i) \times -i$
ASIN(x)	$-i \times LN(\sqrt{1 - x^2} + i \times x)$
ACOS(x)	$\frac{\pi}{2} + i \times LN(\sqrt{1 - x^2} + i \times x)$
ATAN(x)	$-i \times LN(\frac{(1 + i \times x)}{\sqrt{1 + x^2}})$
ASINH(x)	$-LN(\sqrt{1 + x^2} - x)$
ACOSH(x)	$\sqrt{-(\frac{\pi}{2} + i \times LN(\sqrt{1 - x^2} + i \times x))^2}$
ATANH(x)	$-LN(\frac{(1 - x)}{\sqrt{1 - x^2}})$

TRG* Expansions

Function	Expansion
SIN$(x + y)$	$SIN(x) \times COS(y) + COS(x) \times SIN(x)$
COS$(x + y)$	$COS(x) \times COS(y) + SIN(x) \times SIN(y)$
TAN$(x + y)$	$\frac{TAN(x) + TAN(y)}{1 - TAN(x) \times TAN(y)}$
SINH$(x + y)$	$SINH(x) \times COSH(y) + COSH(x) \times SINH(x)$
COSH$(x + y)$	$COSH(x) \times COSH(y) + SINH(x) \times SINH(y)$
TANH$(x + y)$	$\frac{TANH(x) + TANH(y)}{1 + TANH(x) \times TANH(y)}$

→TRG Expansion

Function	Expansion
EXP(x)	$COS(\frac{x}{i}) + SIN(\frac{x}{i}) \times i$

Source References

The following references were used as sources for many of the constants and equations used in the HP 48. (See "References" in chapter 4, "Equation Reference," for the references used as sources for the Equation Library.)

1. E.A. Mechtly. *The International System of Units, Physical Constants and Conversion Factors*, Second Revision. National Aeronautics and Space Administration, Washington DC, 1973.

2. *The American Heritage Dictionary*. Houghton Mifflin Company, Boston, MA, 1979.

3. *American National Standard Metric Practice ANSI/IEEE Std 268-1982*. The Institute of Electrical and Electronics Engineers, Inc., New York, 1982.

4. *ASTM Standard Practice for Use of the International System of Units (SI) E380-89a*. American Society for Testing and Materials, Philadelphia, 1989.

5. *Handbook of Chemistry and Physics*, 64th Edition, 1983-1984. CRC Press, Inc, Boca Raton, FL, 1983.

6. *International Standard publication No. ISO 31/l-1978 (E).*

7. *The International System of Units (SI)*, Fourth Edition. The National Bureau of Standards Special Publication 330, Washington D.C., 1981.

8. *National Aerospace Standard*. Aerospace Industries Association of America, Inc., Washington D.C., 1977.

9. *Physics Letters B*, vol 204, 14 April 1988 (ISSN 0370-2693).

G

Parallel Processing with Lists

Parallel processing is the idea that, generally, if a command can be applied to one or more individual arguments, then it can also be extended to be applied to one or more *sets* of arguments.

Some examples:

- `5 INV` returns `.2`, so `{ 4 5 8 } INV` returns `{ .25 .2 .125 }`.
- `4 5 *` returns `20`, so `{ 4 5 6 } { 5 6 7 } *` returns `{ 20 30 42 }`, and `{ 4 5 6 } 5 *` returns `{ 20 25 30 }`.

General Rules for Parallel Processing

As a rule-of-thumb, a given command can use parallel list processing if all the following are true:

- The command checks for valid argument types. Commands that apply to all object types, such as DUP, SWAP, ROT, and so forth, do not use parallel list processing.
- The command takes exactly one, two, three, four, or five arguments, none of which may itself be a list. Commands that use an indefinite number of arguments (such as →LIST) do not use parallel list processing.
- The command isn't a programming branch command (IF, FOR, CASE, NEXT, and so forth).

The remainder of this appendix describes how the many and various commands available on the HP 48 are grouped with respect to parallel processing.

Group 1: Commands that cannot parallel process

A command must take arguments before it can parallel process, since a zero-argument command (such as RAND, VARS, or REC) has no arguments with which to form a group.

Group 2: Commands that must use DOLIST to parallel process

This group of commands cannot use parallel processing directly, but can be "coerced" into it using the DOLIST command (see "Using DOLIST for Parallel Processing" later in this appendix). This group consists of several subgroups:

- **Stack manipulation commands.** A stack manipulation command cannot parallel process because the stack is manipulated as a whole and list objects are treated the same as any other object. Stack commands (such as DROPN) that take level 1 arguments will not accept level 1 list arguments.

- **Commands that operate on lists as wholes.** Certain commands accept lists as arguments but treat them no differently than any other data object. They perform their function on the object as a whole without respect to its elements. For example, →STR converts the entire list object to a string rather than converting each individual element, and the == command tests the level 1 object against the level 2 object regardless of the objects' types.

- **List manipulation commands.** List manipulation commands will not parallel process since they operate on list arguments as lists rather than as sets of parallel data. However, a list manipulation command can be forced to parallel process lists of lists by using the the DOLIST command. For example, `{ { 1 2 3 } { 4 5 6 } } « ΠLIST » DOLIST` returns `{ 6 120 }`.

- **Other commands that have list arguments.** Because a list can hold any number of objects of any type, it is commonly used to hold a variable number of parameters of various types. Some commands accept such lists, and because of this are insensitive to parallel processing, except by using DOLIST.

- **Index-oriented commands.** Many array commands either establish the size of an array in rows and columns or manipulate individual elements by their row and column indices. These commands expect these row and column indices to be real number pairs collected in lists. For example, `{ 3 4 } RANM` will generate a random integer matrix having 3 rows and 4 columns. Since these commands can normally use lists as arguments, they cannot perform parallel processing, except by using DOLIST.

- **Program control commands.** Program control structures and commands do not perform parallel processing and cannot be forced

to do so. However, programs containing these structures can be made to parallel process by using DOLIST. For example, { 1 2 3 4 5 6 } 1 « IF DUP 3 ≤ THEN DROP END » DOLIST returns { 3 4 5 6 }.

Group 3: Commands that sometimes work with parallel processing

Graphics commands that can take pixel coordinates as arguments expect those coordinates to be presented as two-element lists of binary integers. Since these commands can normally use lists as arguments, they cannot parallel process, except by using DOLIST.

For the two-argument graphics commands (BOX, LINE, TLINE), if either argument is not a list (a complex number, for example), then the commands will parallel process, taking the list argument to be multiple complex number coordinates. For example, (0,0) { (1,1) (3,2) } LINE will draw two lines—between (0,0) and (1,1) and between (0,0) and (3,2).

Group 4: ADD and +

On HP 48S and HP 48SX calculators, the + command has been used to append lists or to append elements to lists. Thus { 1 2 3 } 4 + returns { 1 2 3 4 }. With the advent of parallel processing in the HP 48G series, the ADD command was created to perform parallel addition instead of +.

This has several ramifications:

- To add two lists in parallel, you must do one of the following:
  - Use ADD from the (MTH) LIST menu.
  - Create a custom menu containing the ADD command.
  - Assign the ADD command to a user-defined key.
- User programs must be written using ADD instead of + if the program is to be able to perform direct parallel processing, or written with + and applied to their arguments by using DOLIST. For example, programs such as « →x '×+2' » will produce list concatenation when x is a list rather than parallel addition, unless rewritten as « →x '× ADD 2' »
- Algebraic expressions capable of calculating with variables containing lists (including those intended to become user-defined functions) cannot be created in RPN syntax since using

ADD to add two symbolic arguments concatenates the arguments with + rather than with ADD. For example, 'X' DUP 2 ^ SWAP 4 * ADD 'F(X)' SWAP = produces 'F(X)=X^2+4*X' rather than 'F(X)=X^2 ADD 4*X'.

Group 5: Commands that set modes/states

Commands that store values in system-specific locations so as to control certain modes and machine states can generally be used to parallel process data. The problem is that each successive parameter in the list cancels the setting established by the previous parameter. For example, { 1 2 3 4 5 } FIX is effectively the same as 5 FIX.

Group 6: One-argument, one-result commands

These commands are the easiest to use with parallel processing. Simply provide the command with a list of arguments instead of the expected single argument. Some examples:

{ 1 -2 3 -4 } ABS returns { 1 2 3 4 }

DEG { 0 30 60 90 } SIN returns { 0 .5 .866025403784 1 }

{ 1 A 'SIN(Z)' } INV returns {1 'INV(A)' 'INV(SIN(Z))'

Group 7: Two-argument, one-result commands

Two-argument commands can operate in parallel in any of three different ways:

- { *list* } { *list* }
- { *list* } *object*
- *object* { *list* }

In the first form, parallel elements are combined by the command: { 1 2 3 } { 4 5 6 } % returns { .04 .1 .18 }.

In the second form, the level 1 object is combined with each element in the level 2 list in succession:
{ 1 2 3 } 30 %CH returns { 2900 1400 900 }.

In the third form, the level 2 object is combined with each element of the level 1 list in succession:
50 { 1 2 3 } %T returns { 2 4 6 }.

Group 8: Multiple-argument, one-result commands

Commands that take multiple (3, 4, or 5) arguments can perform parallel processing only if all arguments are lists. For example, { 'SIN(X)' 'COS(X)' 'TAN(X)' } { X X X } { 0 0 0 } ROOT returns { 0 90 0 }. Notice that lists must be used even though the level 1 and level 2 lists each contain multiples of the same element.

Group 9: Multiple-result commands

Any command that allows parallel processing, but produces multiple results from its input data, will return its results as a single list. For example, { 1 2 3 } { 4 5 6 } R→C C→R produces { 1 4 2 5 3 6 } rather than the more expected { 1 2 3 } { 4 5 6 }.

The following *UNMIX* program will unmix the data given the number of expected result lists:

```
« OVER SIZE → l n s
 « 1 n
  FOR j j s
   FOR i l i GET n
   STEP s n / →LIST
  NEXT
 »
»
```

Taking { 1 4 2 5 3 6 } from above as the result of C→R (a command which should return two results), 2 UNMIX gives { 1 2 3 } { 4 5 6 }.

Group 10: Quirky commands

A few commands behave uniquely with respect to parallel processing:

- **DELALARM.** This command can take a list of arguments. Note, however, that deletions from early in the alarm list will change the alarm indices of the later alarm entries. Thus, if there are only three alarms, { 1 3 } DELALARM will cause an error, whereas { 3 1 } DELALARM will not.
- **DOERR.** This command forces an error state that causes all running programs and commands to halt. Thus, even though providing the command with a list argument will cause the command to perform parallel processing, the first error state

will cause the command to abort and none of the rest of the list arguments will be used.

- **FREE, MERGE.** Only port 1 can be freed or merged on the HP 48GX. Thus, even though a list argument is acceptable, an error will occur for any list except { 1 }.
- **RESTORE.** This command performs a system warmstart after installing the backup object into memory. All functions are terminated at that time. Thus, only the first backup object in a list will be restored.
- **_ (Attach Unit).** This command will create unit objects in parallel only if level 1 contains a list. Thus `1 { ft in m } _` produces `{ 1_ft 1_in 1_m }` while `{ 1 2 3 } 'm' _` produces an error.
- **STO+.** STO+ performs parallel list addition only if both arguments are lists. If one argument is a list and the other is not, STO+ appends the non-list argument to each element in the list.
- **STO−, STO*, STO/.** These commands perform parallel processing if both arguments are lists, but fail otherwise.

Using DOLIST for Parallel Processing

Almost any command or user program can be made to work in parallel over a list or lists of data by using the DOLIST command. Use DOLIST as follows.

- Level 1 must contain a command, a program object, or the name of a variable that contains a command or program object.
- Level 2 must contain an argument count unless the level 1 object is a command that accepts parallel processing, a program containing only one command that accepts parallel processing, or a user-defined function. In these special cases, Level 2 contains the first of the list arguments.
- If level 2 was the argument count, then level 3 is the first of the argument lists. Otherwise, levels 2 through n are the argument lists.

As an example, the following program takes three objects from the stack, tags them with the names a, b, and c, and displays them one after the other in line 1 of the display.

```
« → a b c
  « { a b c } DUP « EVAL » DOLIST
    SWAP « →TAG » DOLIST
    CLLCD 1 « 1 DISP 1 WAIT » DOLIST
  »
»
```

Index

Special characters

℗ character, 1-10

A

absolute value, 3-5
alarms
  acknowledging, 3-6
  deleting, 3-78
  finding, 3-116
  index number, 3-116
  recalling, 3-262
  storing, 3-326
ALG annunciator, 1-9
algebraic
  linear structure, 1-20
Algebraic/Program-entry mode, 1-9, 1-63
algebraics
  action in programs, 1-2
  collecting terms, 3-56
  comparing, 1-19
  conditional testing, 1-22
  editing in programs, 1-9
  expanding, 3-109
  in local variable structure, 1-3, 1-11
  isolating variables, 3-158
  rearranging, 3-104, 3-158
  rearranging programmatically, 2-19
  simplifying, 3-56
  testing for linearity, 3-168
  tests in, 1-19
algebraic syntax
  conditional testing, 1-22
  in local variable structures, 1-4
  test commands, 1-17, 1-19
alpha keyboard
  automatically locking, 1-63
amortization (TVM), 3-12
angle mode
  setting, 3-78, 3-131, 3-256
angles
  converting units, 3-99, 3-294
angular mechanics, 4-29
angular motion, 4-48
animation, 2-45, 2-56, 3-14
annunciators
  user flags, 1-42
applications, 1-79
archives
  creating, 3-18
arcs, 3-17
arguments
  comparing, 3-183, 3-191
  recalling last, 3-162, 3-163
  verifying, 2-36
arrays
  applying a program to, 2-29
  column norm, 3-53
  complex conjugates, 3-62
  constant, 3-59

creating from stack, 3-20, 3-369
deleting columns, 3-55
disassembling, 3-19
disassembling complex, 3-71
extracting elements, 3-127, 3-129
Fourier transforms, 3-115
inserting columns, 3-54
inserting diagonals, 3-84
inverse Fourier transform, 3-147
manipulating, 2-16, 2-49
maximum and minimum elements, 2-22
redimensioning, 3-266
replacing elements in, 3-244, 3-246
residual, 3-291
row norm, 3-280
sorting elements, 2-23
spectral norm of, 3-314
swapping columns, 3-70
symbolic, 2-29
axes (plots)
controlling, 3-33
including, 3-95
labeling, 3-162

B

backup objects, 3-248
creating, 3-18
restoring, 3-273
bar graphs, 3-34
base
setting, 3-38, 3-75, 3-137, 3-210
baud rate
specifying, 3-36
beams, 4-1
beeper
in programs, 1-71
sounding, 3-37
specifying tone and duration, 3-37
Bernoulli equation, 4-23
Bessel functions, 2-43
binary integers
comparing, 1-19
converting to floating-point, 3-42
custom display, 2-7
representing flags, 1-44
shifting one bit right, 3-25
wordsize, 1-19, 3-335
binary wordsize
recalling, 3-265
bipolar transistors, 4-73
bit
rotate left, 3-278
rotate right, 3-285
shift left, 3-310
shift right, 3-317
black-body emissive power, 3-112
black body radiation, 4-42
boxes, 3-40
branch cuts, 3-7, 3-9, 3-21, 3-23, 3-26, 3-29
branching structures
conditional structures, 1-20, 1-53, 3-101
ending, 3-102
loop structures, 1-27
program element, 1-3
BRCH menu, 1-20, 1-27
Brewster angle, 4-52
buckling, 4-3
buffer (serial)
clearing, 3-51
sizing data in, 3-40

byte
rotate left, 3-278
rotate right, 3-285
shift left, 3-311
shift right, 3-318

C

calculator
turning off, 1-82
calculator clock, 2-5
cantilevers, 4-1
capacitor, 4-16, 4-17, 4-19, 4-20
"case" branching, 1-22, 2-38, 2-49, 3-42
case structures, 2-49
centripetal force, 4-29
character codes
remapping to match HP 82240A, 3-211
to characters, 3-48
characters
codes, 3-204
character translation, 3-352
checksums, 3-41
specifying type of, 3-49
verify programs, 2-1
chi-square distribution, 3-363
choose boxes
custom, 1-68, 3-46
in programs, 1-68
circle, 4-59, 4-79
circular motion, 4-49
clearing
command line, 3-50
directories, 3-53
display, 1-74
flags, 1-42, 3-45
stack, 3-50
stack display, 3-51
subdirectories, 3-52
variables, 3-52, 3-53
clock
adjusting, 3-50
coefficients
of monic polynomials, 3-220
regression, 3-177
collisions, 4-30
column operations
converting matrices into columns, 3-54
converting vectors into elements, 3-54
creating matrices from columns, 3-56
creating vectors from elements, 3-56
deleting, 3-55
inserting, 3-54
columns, 4-1
combinations, 3-58
command line
clearing, 3-50
during program input, 1-62
commands
applying to list elements, 3-92, 3-102, 3-203
applying to lists, 3-91
in programs, 1-2
comments, 1-10
comparing objects, 3-294
comparison functions, 1-17, 1-19
compiled local variable structures
defining procedure, 1-15
complex numbers
conjugates, 3-62
disassembling, 3-71
imaginary parts, 3-150
polar angle θ, 3-19
real parts, 3-267
computer
creating programs on, 1-10

conditional commands, 1-20, 1-21, 1-22
conditionals
  nested, 2-23, 2-26, 2-36
conditional structures
  "case" branching, 1-22, 3-42
  conditional commands, 1-20
  ending, 3-102
  error branching, 1-53, 1-54, 3-146
  examples, 1-23
  "if" branching, 1-20, 1-21, 1-22, 1-53, 1-54, 3-101, 3-144, 3-148, 3-149
  program element, 1-3
  test commands in, 1-17, 1-20
conduction, 4-39, 4-41
cone, 4-64
conjugates, 3-62
  of matrices, 3-353
  of objects, 3-300
constants
  symbolic, 3-99, 3-143, 3-183, 3-193
Constants Library
  opening, 3-63
continuing execution, 1-47, 1-48, 1-57, 1-59, 3-64
convection, 4-40, 4-41
converting base units, 3-360
coordinate modes
  specifying, 3-70, 3-269, 3-316
coordinates
  pixels to user units, 3-250
  specifying for *PICT*, 3-231
  user units to pixels, 3-71
coordinates of PICT
  specifying, 3-231
correlation (statistical), 3-65
Coulomb's law, 4-11
counters
  loop structures, 1-29, 1-31, 1-33, 1-35
  negative steps, 1-31, 1-35, 1-39
  stepping, 1-39
covariance, 3-222
creating 2D vectors, 3-369
creating 3D vectors, 3-370
critical angle, 4-51
current, 4-9, 4-43, 4-67
cursor (command line), 1-63
curve fitting, 3-37, 3-110, 3-166, 3-167, 3-176, 3-250
custom menus
  creating, 3-187
  displaying, 3-187
  in programs, 1-78, 1-79
  menu-based applications, 1-79
cylinder, 4-64

D

Darcy friction factor, 3-72
data input, 3-154
data transmission
  closing ports, 3-51
  detecting errors, 3-49
  error testing, 3-40
  parity, 3-217
  receiving, 3-268, 3-269
  serial, 3-381
  size of, 3-40
  specifying baud rate, 3-36
  terminating Server mode, 3-117
  time-out, 3-324
  via Kermit, 3-160, 3-301
  via Kermit server, 3-160, 3-303

data transmissions
opening ports, 3-212
dates
calculating days between, 3-75
calculating past or future, 3-74
displaying, 3-73, 3-356
setting, 3-73
debugging, 1-47, 1-49, 3-74, 3-201
decomposing vectors, 3-371
defining procedures
compiled local variables in, 1-15
local variables in, 1-14
local variable structures, 1-11
definite loops, 2-3, 2-26, 2-46, 2-47, 2-49, 2-56
with counters, 2-10, 2-16
delimiters
« » for programs, 1-1
dependent variables
specifying in matrices, 3-389
specifying in plots, 3-81
specifying in statistical data, 3-57
summation of squares, 3-389
summations of, 3-388
dialog boxes (input forms), 3-152
dimensions
converting, 3-65
of PICT, 3-223
diodes, 4-69
directories
changing, 3-142, 3-363
clearing, 3-53
creating, 3-69
current, 3-142
HOME, 3-142
paths, 3-219
purging, 3-225
display
area numbers, 1-59
clearing, 1-74
clearing stack, 3-51
creating graphics objects from, 3-163
freezing, 1-58, 3-123
printing, 3-237
display mode
setting, 3-103, 3-117, 3-299, 3-322
"do" looping, 1-36, 2-19, 2-23, 2-43, 3-89, 3-362
drag force, 4-31

E

editing
programs, 1-9
eigenvalues
of matrices, 3-100, 3-101
eigenvectors
of matrices, 3-100
elastic buckling, 4-3
elastic collisions, 4-30
electricity, 4-9
electrostatic force, 4-11
ellipse, 4-59
energy, 4-15, 4-31
Equation Library
references, 4-1, 4-82
starting, 3-104
subjects, 4-1
titles, 4-1
equations
defining sets, 3-193
disassembling, 3-104
expanding, 3-109
least squares solution, 3-178
rearranging, 3-104

recalling, 3-259
reordering sets, 3-194
retitling sets, 3-194
sets, 3-194
solving linear systems, 3-178
solving quadratic, 3-253
solving sets, 3-197
testing for linearity, 3-168
equation sets
changing titles, 3-194
defining, 3-193
reordering, 3-194
solving, 3-197
errors
actions in programs, 1-51
analyzing, 1-51
causes, 1-51
causing, 1-51, 3-90
clearing last, 1-51, 3-106
conditional structures, 1-53, 1-54, 3-146
detecting in transmission, 3-49
display messages, 1-51
Kermit, 3-159
numbers for, 1-51, 3-106
recalling messages, 1-51, 3-105, 3-159
trapping, 1-53, 1-54, 3-146
user-defined, 1-51
error-trapping, 2-29
error trapping, 2-9, 2-10
escape velocity, 4-49
evaluation
of local variables, 1-13
of test clauses, 1-21, 1-22, 1-36, 1-38
example program
UNMIX, G-5
example programs
animating graphics, 2-45, 2-47, 2-56
applying programs repeatedly, 2-19
Bessel functions, 2-43
calculating median, 2-14
converting from algebraic to RPN, 2-40
converting plots to grobs, 2-45
custom menus, 1-80
displaying binary integers, 2-10
execution times, 2-5
Fibonacci numbers, 2-2
input forms, 1-67
input routines, 1-54, 1-57, 1-60, 1-64, 1-66
inverse functions, 2-54
manipulating math curves, 2-46
mass of an object, 1-79
maximum and minimum elements, 2-22
percentile of a list, 2-14
phone list, 1-67
plotting pie charts, 2-49
preserving calculator status, 2-8
rearranging algebraics, 2-19
right-justifying strings, 2-7
summations, 1-41
surface area of a torus, 1-45
system flags, 1-43
Taylor's polynomials, 2-45
trace mode, 2-53
using conditionals, 1-23, 1-24, 1-25, 1-26
using loops, 1-29, 1-31, 1-33, 1-35, 1-37, 1-40

verifying arguments, 2-36
volume of a sphere, 1-5, 1-6
volume of a spherical cap, 1-6, 1-9, 1-12
Examples subdirectory
creating, 3-345
removing, 3-52
exponents, 3-383
expressions
creating from arguments, 3-15
pattern replacement, 3-180, 3-181
rewriting, 3-180, 3-181
extrapolation, 3-234, 3-235, 3-236

F

factorials, 3-112
factor units, 3-361
false (test result), 1-17, 1-19
Fanning friction factor, 3-113
F distribution, 3-364
Fibonacci numbers, 2-2
flags
annunciators, 1-42
binary integer form, 1-44
clearing, 1-42, 3-45, 3-114, 3-125
control behavior, 1-42
controlling logic with, 2-23, 2-26
default states, C-1
preserving and restoring status, 2-8, 2-9, 2-49
program control, 1-42
recalling, 3-262
recalling states, 1-44
restoring states, 1-44
setting, 1-42, 2-10, 2-23, 2-26, 2-53, 3-304, 3-327
storing states, 1-44
system, 1-42, 1-44, C-1
testing, 1-42, 2-23, 2-26, 3-114, 3-124, 3-125
testing and clearing, 3-114, 3-125
types, 1-42
user, 1-42, 1-44
fluid flows
Darcy friction factor, 3-72
Fanning friction factor, 3-113
fluids, 4-22
focal length, 4-50
force, 4-11, 4-27, 4-44
"for" looping, 1-32, 1-34, 2-10, 2-16, 2-26, 2-46, 2-47, 2-49, 2-56, 3-119, 3-201
formatting
ports, 3-228
Fourier transforms
inverse, 3-147
of arrays, 3-115, 3-147
free fall motion, 4-47
freeing merged memory, 3-121, 3-122
frequency
resonant, 4-19
friction losses, 4-26
functions
applying to list elements, 3-92, 3-102, 3-203
applying to lists, 3-91
defining, 3-77

G

gamma function, 3-112
gas compressibility correction factor, 3-393
gas-compressibility factor, 4-36
gases, 4-32
geometry, 4-58, 4-63

global variables
action in programs, 1-2
disadvantages in programs, 1-11
list of, 3-357
graphics
creating, 3-17, 3-40, 3-131, 3-166, 3-227
custom, 2-56
displaying, 3-249
environment, 3-131
Picture environment, 3-227
graphics commands
parallel processing with, G-3
graphics objects
animating, 3-14
creating blank, 3-39
creating from display, 3-163
creating from stack, 3-133
displaying, 3-164
manipulating, 2-23, 2-46, 2-47, 2-49, 2-56
superimposing, 3-130, 3-134
gravitation, 4-31, 4-47, 4-48, 4-49

H

HALT annunciator, 1-47, 1-59
halting programs, 1-48, 3-136
harmonic motion, 4-54
head loss, 4-24
heat, 4-37
heat capacity, 4-38
heat transfer, 4-37
histograms, 3-137, 3-139
HMS format
adding in, 3-139
converting from, 3-141
converting to, 3-142
subtracting in, 3-140
Hooke's law, 4-30

I

ideal gases, 4-32
"if" branching, 1-20, 1-21, 1-22, 1-53, 1-54, 2-2, 2-23, 2-26, 2-36, 3-101, 3-144, 3-148, 3-149, 3-346
"iferror" branching, 2-29
implicit variable references, 3-305
indefinite loops, 2-7, 2-19, 2-23
ending, 3-89, 3-102, 3-374
with counters, 2-43
independent variables
specifying for plotting, 3-151
specifying in matrices, 3-380
specifying in statistical data, 3-57
summation of squares, 3-379
summations of, 3-379, 3-388
index of refraction, 4-50
inductor, 4-17, 4-20, 4-21
initial value problems, 3-274
error estimate, 3-276, 3-290
next step, 3-288
solution step size, 3-277
with known partials, 3-286
input
prompting for, 3-238
input forms
creating, 3-152
custom, 1-67, 3-152
for program input, 1-67
in programs, 1-67
resetting, 3-203
saving initial values, 3-203
input plane
setting the x range, 3-387
setting the y range, 3-393
intrinsic density of silicon, 3-305

inverses
calculating, 3-156
logical, 3-201
of matrices, 3-156
storing, 3-309
I/O
closing ports, 3-51
IR port, 3-157
Kermit errors, 3-159
Kermit transmission, 3-160
selecting port, 3-157
serial port, 3-157
isothermal expansion, 4-34

J

junction field-effect transistors, 4-74

K

Kermit, 3-212, 3-217, 3-230, 3-268, 3-269, 3-301, 3-303
error messages, 3-159
server, 3-160
transmission, 3-160
keyboard
defining user, 3-24, 3-328
in programs, 1-72, 1-73
recalling definitions, 3-263
unassigning user keys, 3-80
key location numbers, 1-73
keys
defining in user keyboard, 3-24
testing, 3-159
keystrokes
as program input, 1-72, 1-73
waiting for , 3-373
killing programs, 1-47, 1-48, 3-161

L

last argument
recalling, 2-10, 3-162, 3-163
length factor, 4-3
libraries
attaching, 3-31
detaching, 3-84
displaying menus, 3-187
listing, 3-165
light, 4-50
linear mechanics, 4-28
linear motion, 4-47
linear structure, 1-20
lines, 3-166
list concatenation, G-3
list processing
programming example, 2-29
lists
action in programs, 1-2
adding elements of two, 3-11
applying commands, functions, or programs to, 3-91, 3-92, 3-102, 3-203, 3-302, 3-333, 3-336
applying executable object repeatedly, 3-302
assembling, 3-169
creating from stack, 3-169
differences between elements, 3-170
disassembling, 3-169
extracting elements, 3-127, 3-129, 3-136, 3-341
first element, 3-136
last elements, 3-341
locating objects in, 3-234
multiplying elements of, 3-171
parallel processing, G-1
product of, 3-171
replacing elements in, 3-244, 3-246

reversing element order, 3-274
sorting, 2-14, 3-315
sublist position in, 3-203
summing elements of, 3-170
local variables, 2-9
action in programs, 1-2
compiled, 1-15
creating, 1-3, 1-11
evaluating, 1-13, 2-19
exist temporarily, 1-11, 1-12, 1-14
naming, 1-11
nested, 2-43, 2-49
passing between programs, 2-47
storing objects in, 2-19, 2-38
local variable structures
advantages, 1-12
as user-defined functions, 1-16
calculations with, 1-3
create local variables, 1-11
defining procedure, 1-11, 1-14
entering, 1-11
operation, 1-3, 1-11
program element, 1-3
syntax, 1-3, 1-11
logic
controlling, 2-26
controlling with flags, 2-23, 2-26
functions, 2-23, 2-26, 2-36, 2-38
logical functions, 1-17, 1-19, 3-13, 3-201, 3-213, 3-382
longitudinal waves, 4-81
loop structures
counters, 1-29, 1-31, 1-33, 1-35, 1-39, 3-323
definite, 1-27, 1-28, 2-3, 2-26, 2-46, 2-47, 2-49, 2-56, 3-119, 3-201, 3-321
"do" looping, 1-36, 3-89
"for" looping, 1-32, 1-34, 3-119, 3-201, 3-323
indefinite, 1-27, 1-36, 2-7, 2-19, 2-23, 3-374
keystroke input, 1-73
negative steps, 1-31, 1-35
program element, 1-3
"start" looping, 1-28, 1-30, 3-201, 3-321
summation alternative, 1-40
test commands in, 1-36, 1-38
"while" looping, 1-38
lowercase letters
in names, 1-11

M

Mach number, 4-35
magnetic field, 4-15, 4-43, 4-44, 4-45
magnetism, 4-43
magnification, 4-50
mantissas, 3-179
mass
related to energy, 4-31
mathematical data
plotting, 3-94
matrices
adding rows, 3-283
condition number, 3-60
conjugates of, 3-353
converting to columns, 3-54
converting to rows, 3-282
creating from columns, 3-56
creating from rows, 3-284
deleting rows, 3-284
determinants, 3-82
eigenvalues, 3-101

eigenvalues and eigenvectors, 3-100
extracting diagonals, 3-85
identity, 3-143
inverting, 3-156
least squares solution, 3-178
LQ factorization, 3-176
LU decomposition, 3-179
multiplying rows by a constant, 3-260
QR factorization, 3-253
random element, 3-257
rank of, 3-257
reduced row echelon form, 3-286
Schur decomposition, 3-298
singular value decomposition, 3-337
singular values of, 3-338
spectral radius, 3-317
squaring, 3-316
sum of diagonal elements, 3-351
swapping rows, 3-292
transposing, 3-353
mechanics, 4-28, 4-29
memory
checking available, 3-186
freeing merged, 3-121
merging RAM card, 3-190
menu-based applications, 1-79
menu descriptions
PRG BRCH, 1-20, 1-27
PRG RUN, 1-49
PRG TEST, 1-18
menus
custom, 1-78, 1-79, 2-23, 3-187
defining, 3-187
delayed display, 1-72, 1-78
displaying, 3-187
displaying in programs, 1-72, 1-77, 1-79
for libraries, 1-77
for program input, 1-79
last menu, 1-78
numbers for, 1-77, 1-78, 3-264
pages in, 1-78
programmatic uses, 1-77
recalling, 3-264
recalling numbers, 1-78
resuming programs, 1-79
running programs, 1-79
temporary, 2-49, 3-350
message boxes
creating, 3-196
custom, 1-77, 3-196
in programs, 1-77
messages, A-1–17
displaying, 3-88
prompting, 1-56
meta-objects, 2-29
Minehunt game, 3-192
cheating, 3-192
storing, 3-192
mode names
Algebraic-entry, 1-63
Algebraic/Program-entry, 1-9, 1-63
Program-entry, 1-4, 1-9
modes
program entry, 1-4, 1-9
setting, 1-9
Mohr's circle, 4-79
motion, 4-46

N

names
action in programs, 1-2
negatives, 3-199
nested structures, 2-40, 2-43
new commands, E-1–7

newlines, 1-4, 1-59
NMOS transistors, 4-71
nonideal hydrocarbon gas, 3-393
normal distribution, 3-365
NPN bipolar transistors, 4-73
number bases
  converting between, 2-32
numbers
  action in programs, 1-2
  complex, 3-19
  complex conjugates, 3-62
  disassembling complex, 3-71
  fractional part, 3-121
  imaginary parts, 3-150
  integer part, 3-157
  largest available, 3-183
  rational form, 3-251
  rational form with π, 3-251
  real parts, 3-267
  real to binary, 3-292
  real to complex, 3-293
  rounding, 3-279
  rounding to integer, 3-44, 3-118
  smallest available, 3-193
  squaring, 3-316

O

objects
  actions in programs, 1-2
  backup, 3-248
  comparing, 3-294
  conjugates, 3-300
  converting dimensions, 3-65
  copying, 3-200, 3-226
  decomposing, 3-209
  displaying, 3-88
  duplicating, 3-97, 3-98
  entering in programs, 1-4
  evaluating, 3-107
  evaluating by addresses, 3-339
  evaluating symbolic, 3-207
  operating system, 3-339
  printing, 3-241
  recalling, 3-261
  removing from stack, 3-95, 3-96
  removing labels from, 3-97
  removing pointers to, 3-200
  replacing a portion of, 3-270
  sign of, 3-306
  size of, 3-41, 3-309
  storing, 3-325
  storing in reserved variables, 3-324
  storing objects in, 3-325
  testing types, 1-20
  truncating, 3-353
  type numbers, 1-20
  type numbers of, 3-359
  unevaluated, 3-254
object type numbers, 1-20
Ohm's law, 4-11
optics, 4-50
oscillations, 4-54
output
  labeling, 2-5

P

packets
  sending, 3-230
parallel addition, G-3
parallelepiped, 4-65
parallel processing, G-1
  DOLIST, G-2, G-6
  multiple-result commands, G-5
parity
  setting, 3-217
pendulum, 4-56, 4-57
percent functions, 3-47, 3-339
permutations, 3-224

phase delay, 4-16
PICT, 3-226
clearing, 3-105
editing, 3-17, 3-40, 3-166
specifying coordinates, 3-231
superimposing grobs onto, 3-130, 3-134
Picture environment
selecting, 3-131, 3-227
pie charts, 2-49
pixels
checking if on, 3-229
coordinates, 3-71, 3-250
toggling, 3-349
turning off, 3-228
turning on, 3-229
plane geometry, 4-58
plots
3D, 3-207, 3-208
3D perspective, 3-111
autoscaling, 3-32
center, 3-44
changing horizontal width, 3-372
controlling axes, 3-33
creating, 3-36
drawing, 3-94, 3-139, 3-296
drawing axes, 3-95
labeling axes, 3-162
mathematical data, 3-94
resolution, 3-207, 3-208, 3-271
scaling, 3-135, 3-299
setting axes tick-marks, 3-30
setting types, 3-34, 3-61, 3-86, 3-125, 3-132, 3-137, 3-215, 3-296, 3-297, 3-355, 3-375, 3-391
simultaneous, 3-307
specifying dependent variable, 3-81
specifying eye point, 3-111
specifying x-axis display range, 3-384
specifying y-axis display range, 3-390
statistical data, 3-36, 3-94, 3-137, 3-139
plot types
setting, 3-218, 3-221, 3-312
specifying, 3-232
plug-in cards
initializing, 3-228
PN step-junction devices, 4-69
polarization, 4-52
polygon, 4-61
polynomials
evaluating, 3-224
monic, 3-220
roots, 3-238
Taylor's, 3-343
polytropic processes, 4-34
population covariance, 3-222
population standard deviation, 3-242
population variance, 3-247
port
opening, 3-212
selecting, 3-157
ports
closing, 3-51
initializing, 3-228
pressure
hydrostatic, 4-23
PRG annunciator, 1-4, 1-9
PRG BRCH menu, 1-20, 1-27
PRG RUN menu, 1-49
PRG TEST menu, 1-18
print buffer
printing, 3-68
printing
HP 82240A, 3-211
print buffer, 3-68

setting delay, 3-79
trace mode, 2-53
Program Development Link, 1-10, 2-1
programs included, 2-1
Program-entry mode, 1-4, 1-9
program examples
arbitrary number bases, 2-32
programming techniques
applied list processing, 2-29
"case" branches, 2-38, 2-49
case structures, 2-38, 2-49
controlling logic with flags, 2-23, 2-26
custom graphics, 2-56
custom menus, 2-23, 2-26
definite loops, 2-3, 2-26, 2-46, 2-47, 2-49, 2-56
definite loops with counters, 2-10, 2-16
"do" loops, 2-19, 2-23, 2-43
error-trapping, 2-29
error trapping, 2-9, 2-10
evaluating local variables, 2-19
"for" loops, 2-10, 2-16, 2-26, 2-46, 2-47, 2-49, 2-56
"if" branches, 2-23, 2-26
indefinite looping, 2-33
indefinite loops, 2-7, 2-19, 2-23
indefinite loops with counters, 2-43
interpolation, 2-14
labeling output, 2-5
list concatenation, 2-40
local variables, 2-9, 2-38, 2-43, 2-47, 2-49
logical functions, 2-23, 2-26, 2-36, 2-38
logic control, 2-26
manipulating grobs, 2-23, 2-46, 2-47, 2-49, 2-56
meta-object manipulation, 2-29
nested conditionals, 2-23, 2-26, 2-36
nested structures, 2-40, 2-43
object type-checking, 2-40
plot commands, 2-45, 2-46, 2-49
preserving flag status, 2-9, 2-49
programs as arguments, 2-10, 2-19, 2-54
recursion, 2-2, 2-40
restoring flag status, 2-9, 2-49
restoring last argument, 2-10
root-finder, 2-54
setting flags, 2-10, 2-23, 2-26, 2-53
simulating new object types, 2-29
sorting array elements, 2-23
sorting lists, 2-14
"start" loops, 2-3
string and character manipulation, 2-33
string operations, 2-7
structures, 2-5
subroutines, 2-5, 2-10, 2-21, 2-38
tagged output, 2-33
temporary menus, 2-49
testing flags, 2-23, 2-26
using arrays, 2-16, 2-49
using calculator clock, 2-5
using flags, 2-29
using other programs, 2-5, 2-10, 2-21, 2-38

using statistics commands, 2-49
utility programs, 2-38
vectored enter, 2-53
"while" loops, 2-7
programs
actions for object types, 1-2
applying to elements of a matrix, 2-29
applying to list elements, 3-92, 3-102, 3-203
applying to lists, 3-91
are sequences of objects, 1-1
beeping, 1-71
calculation styles, 1-3
cancelling halted, 3-161
causing errors, 1-51
checksums, 2-1
comments in, 1-10
conditional structures, 1-20, 1-53, 1-54
creating on computer, 1-10
cursor position during input, 1-63
debugging, 1-47, 3-74, 3-201
default input, 1-60
displaying input forms, 1-67
displaying menus, 1-72, 1-77, 1-79, 2-23, 2-26, 2-49
displaying output, 1-74, 1-75, 1-76
displaying string output, 1-75
editing, 1-9
elapsed time, 2-5
entering, 1-4
entry modes, 1-4, 1-9
entry modes during input, 1-63
error actions, 1-51
evaluating local variables, 1-13
executing, 1-5
finding roots in, 2-54
flags in, 1-42
getting input, 1-55, 1-56, 1-58, 1-60, 1-72, 1-73, 3-154
HALT annunciator, 1-47
halting, 1-48, 3-136
in local variable structure, 1-3, 1-11
input as strings, 1-60
input forms, 3-152
introduction, 1-1
killing, 1-47, 1-48, 3-161
labeling output, 1-74, 1-75
local variable structures, 1-3, 1-11
loop structures, 1-27, 3-89, 3-374
naming, 1-5
newlines in, 1-4
not evaluating local variables, 1-13
not executing in programs, 1-2
objects in, 1-2
on the stack, 1-4
pausing, 3-373
pausing for output, 1-76
prompting, 1-56, 1-58, 1-60, 3-154
recursion, 2-2
resuming, 1-47, 1-48, 1-56, 1-59, 1-79, 1-82, 3-64
samples, 3-345
scope of local variables in, 1-14, 1-15
single-step execution, 1-47, 1-48, 1-49, 3-320
size of, 2-1
stepping through, 3-201

stopping, 1-5, 3-102, 3-161
storing, 1-5
structures in, 1-3
subroutines, 1-45
test commands, 1-17
trapping errors, 1-53, 1-54, 3-102
turning off calculator, 1-82
user-defined functions, 1-16
using as arguments, 2-10, 2-19, 2-54
using as subroutines, 2-5, 2-10, 2-21, 2-38
utility, 2-38
verifying, 2-1
verifying input, 1-63, 2-36
viewing, 1-9
waiting for keystrokes, 1-72, 1-73, 3-373
projectile motion, 4-48
prompting, 1-56, 1-58, 1-60
prompts, 3-154

Q

quadratic equations
solving, 3-253
quality factor, 4-19

R

RAM
checking available, 3-186
RAM cards
freeing, 3-121, 3-122
merging, 3-190
random numbers
generating, 3-256
in matrices, 3-257
seeding, 3-267
reactance, 4-16
real gases, 4-32
real numbers
converting to binary, 3-292
converting to complex, 3-293
manipulating, 2-33
recalling
flag states, 1-44
last arguments, 3-163
menu numbers, 1-78
rectangle, 4-60
recursion, 2-2, 2-40
reduced row echelon, 3-286
reflection, 4-52
refraction, 4-50
regression
calculating, 3-177
formula used, 3-166
power, 3-250
setting type, 3-37, 3-110, 3-167, 3-176
remainders, 3-195
reserved variables
storing objects in, 3-324
resistance
wire, 4-13
resonant frequency, 4-19
ring, 4-61
root-finder
in programs, 2-54
roots
finding, 3-281
in programs, 2-54, 3-385
of polynomials, 3-238
row operations
adding rows, 3-283
converting rows to a matrix, 3-284
deleting rows, 3-284
multiplying and adding to another row, 3-260
multiplying by a constant, 3-260

norms, 3-280
swapping rows, 3-292
RPN syntax
converting to, 2-40
Runge-Kutta-Fehlberg, 3-274
RUN menu, 1-49

S

sample standard deviation, 3-300
sample variance, 3-367
Schur decomposition, 3-298
sequential calculations, 3-302, 3-336
serial input buffer, 3-318
serial interrupt, 3-295
serial transmissions, 3-381
input buffer, 3-318
interrupting, 3-295
Server mode
terminating, 3-117
silicon
intrinsic density of, 3-305
silicon devices, 4-67
single-step execution, 1-47, 1-48, 1-49
sinusoidal signal, 4-21
SI units, 3-360
size
binary wordsize, 3-265
initial value solution step, 3-277
of data transmissions, 3-40
of objects, 3-41, 3-309
of programs, 2-1
of stack, 3-82
software
version and date, 3-368
solenoid, 4-20, 4-44
solid geometry, 4-63
solid state devices, 4-67
solver
starting, 3-314
sound waves, 4-81
spectral norm, 3-314
spectral radius of a matrix, 3-317
sphere, 4-66
spring, 4-30, 4-55
squaring, 3-316
stack
calculations on, 1-3
clearing, 3-50
displaying, 3-345
duplicating objects in, 3-97, 3-98, 3-215
manipulating, 3-280, 3-281, 3-282, 3-338
printing, 3-239, 3-240, 3-241
removing objects from, 3-95, 3-96
selecting objects from, 3-226
size of, 3-82
stack display
clearing, 3-51
stack elements to vectors, 3-370
stack syntax
in local variable structures, 1-4
test commands, 1-17
standard deviation
population, 3-242
sample, 3-300
"start" looping, 1-28, 1-30, 2-3, 3-201
state change, 4-33, 4-36
statistical data
chi-square distribution, 3-363
clearing, 3-52
correlation, 3-65, 3-177
covariance, 3-222
extrapolating X, 3-235

extrapolating Y, 3-234, 3-236
F distribution, 3-364
maxima, 3-184
mean, 3-185, 3-198
minima, 3-194
normal distribution, 3-198, 3-365
plotting, 3-36, 3-94, 3-137, 3-139, 3-296
population covariance, 3-222
population standard deviation, 3-242
population variance, 3-247
recalling, 3-264
regression, 3-166, 3-177
sample covariance, 3-67
sample standard deviation, 3-300
sample variance, 3-367
sorting by frequency, 3-38
specifying dependent variable, 3-389
specifying independent and dependent variables, 3-57
specifying independent variable, 3-380
stored in ΣDAT, 3-208
storing, 3-332
summing, 3-351
summing dependent variables, 3-388
summing independent variables, 3-379
summing products of variables, 3-388
summing squared dependent variable, 3-389
summing squared independent variables, 3-379
t distribution, 3-365
upper chi-square distribution, 3-363
upper normal distribution, 3-365
upper Snedecor's F distribution, 3-364
upper students t distribution, 3-365
variance, 3-198
step junction, 4-69, 4-74
storing
flag states, 1-44
programs, 1-5
strain, 4-76
stress, 4-1, 4-76
strings
action in programs, 1-2
as program output, 1-75
concatenation, 2-33
evaluating, 3-333
first characters, 3-204
input converted to, 1-60, 3-334
locating elements in, 3-234
manipulating, 2-7
subdirectories
clearing, 3-52
sublists
number used with DOSUBS, 3-102
subroutines, 2-5, 2-10, 2-21, 2-38
debugging, 1-49
in programs, 1-45
operation, 1-45
single-step execution, 1-49, 3-320
summations
alternative to looping, 1-40
of dependent variables, 3-388

of independent variables, 3-379
of products of statistical variables, 3-388
of squared dependent variables, 3-389
of squared independent variables, 3-379
of statistical data, 3-351
symbolic arrays, 2-29
symbolic constants, 3-99, 3-143, 3-183, 3-193
evaluating, 3-63
symbolic objects
evaluating, 3-207
system time, 3-346, 3-347
setting, 3-347

T

tagged objects
as program output, 1-74
creating, 3-341
Taylor's polynomials
graphing, 2-45
t distribution, 3-365
temperature
change, 3-344
increment, 3-348
terminal velocity, 4-49
test commands
algebraic syntax, 1-17
combining results, 1-19
comparison functions, 1-17
flag tests, 1-42
in conditional structures, 1-17, 1-20
in loop structures, 1-36, 1-38
logical functions, 1-19
results of, 1-17, 1-18, 3-159
stack syntax, 1-17
testing
algebraics, 1-19
binary integers, 1-19
flag states, 1-42
testing linear structure, 1-20
TEST menu, 1-18
thermal expansion, 4-39
tick-marks
setting in plots, 3-30
time, 3-347
and date, 3-356
setting, 3-347
time-out
during data transmission, 3-324
toroid, 4-21, 4-45
trace mode, 2-53
transistors, 4-67
transverse waves, 4-80
trapping errors, 1-51
triangle, 4-62
true (test result), 1-17, 1-19
turning off the calculator, 3-211
TVM, 3-357
begin mode, 3-358
end mode, 3-358
solving, 3-358

U

units
converting angular, 3-99, 3-294
creating from stack, 3-362
factoring, 3-361
numeric portion, 3-366
SI, 3-360
upper chi-square distribution, 3-363
upper normal distribution, 3-365

upper Snedecor's F distribution, 3-364
upper students t distribution, 3-365
user-defined errors, 1-51
user-defined functions
  internal structure, 1-16
user keyboard
  defining keys, 3-24
user keys
  assigning, 3-24
  unassigning, 3-80
utility programs, 2-38

V

variables
  action in programs, 1-2
  adding objects to, 3-329
  clearing, 3-52, 3-53
  decrementing, 1-39, 3-76
  defining, 3-77
  dependent, 3-57, 3-388, 3-389
  designated as calculated, 3-185
  designating user-defined, 3-197
  dividing by, 3-331
  incrementing, 1-39, 3-150
  independent, 3-57, 3-151, 3-379, 3-380, 3-388
  isolating, 3-158
  list of currently used, 3-367
  list of particular type, 3-357
  multiplying, 3-330
  negating, 3-313
  ordering, 3-214
  picture, 3-226
  port, 3-248
  printing, 3-240
  purging, 3-243
  showing, 3-305
  solving multiple, 3-195
  storing objects in, 3-325
  subtracting objects from, 3-330
  types, 3-368
variable types, 3-357
variance, 3-198
  population, 3-247
  sample, 3-367
vectored enter, 2-53
vectors
  converting from matrices, 3-54
  converting to elements, 3-54
  creating from elements, 3-56
  creating from stack, 3-369
  from stack elements, 3-370
  to stack elements, 3-371
vibrations, 4-80
view volume
  setting the depth, 3-392
  setting the height, 3-394
  setting the width, 3-386
voltage, 4-9, 4-67
Vroom, Fruit of the, 2-52

W

waiting
  displaying output, 1-76
  for keystrokes, 1-72, 1-73
warmstart log, 3-377
waves, 4-80
"while" looping, 1-38, 2-7, 3-270, 3-374
while loops, 3-374
wordsize (binary)
  testing, 1-19
wordsize of binary integers, 3-335

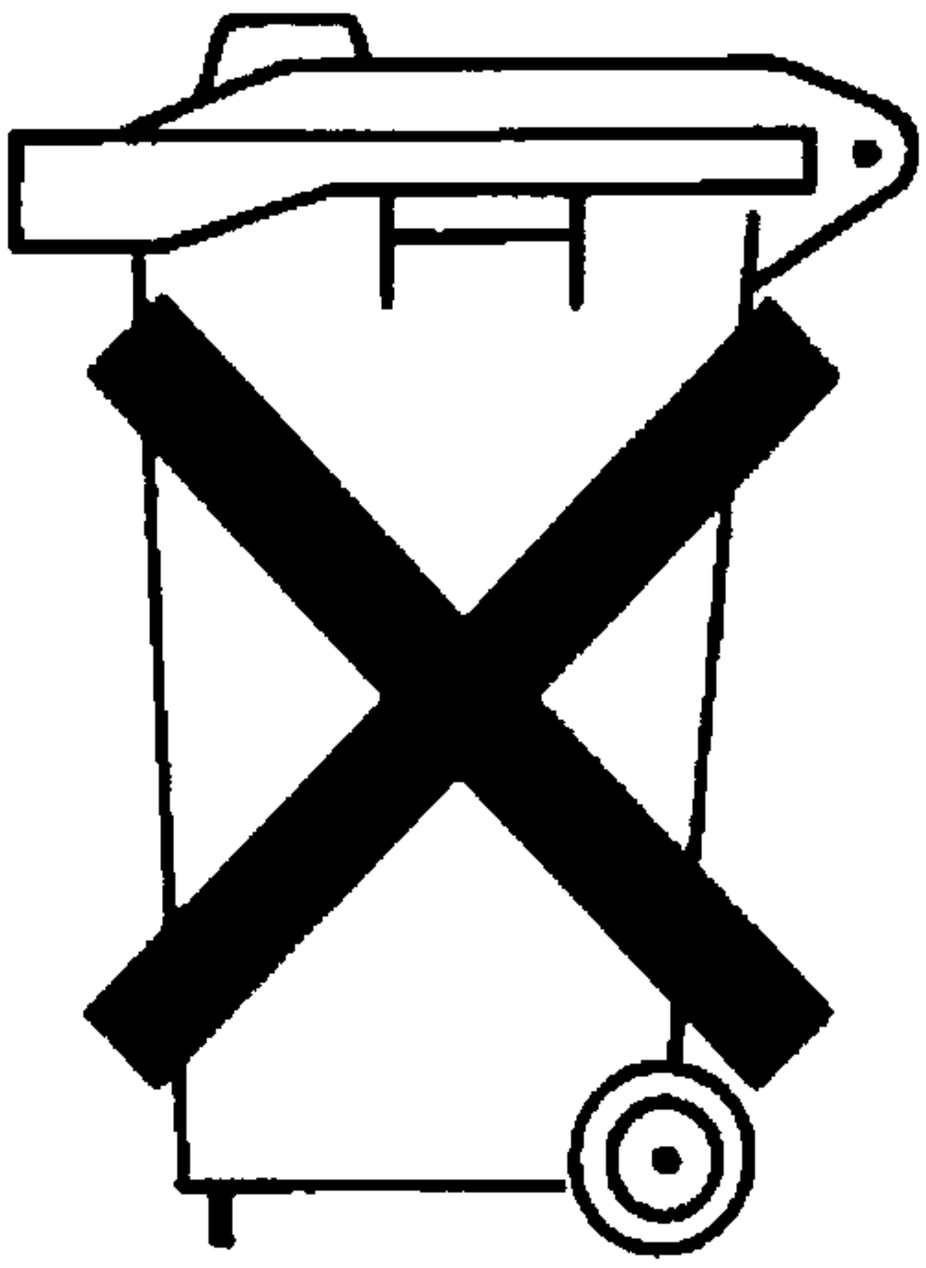
X

x-axis
  specifying display range, 3-384
Xmodem
  receiving objects, 3-384
  sending objects, 3-386

Y

y-axis
  specifying display range, 3-390

This regulation applies only to The Netherlands



Batteries are delivered with this product, when empty do not throw them away but collect as small chemical waste.

Bij dit produkt zijn batterijen geleverd. Wanneer deze leeg zijn, moet u ze niet weggooien maar inleveren als KCA.